Volume 18 Number 4

ACTA CYBERNETICA

Szeged, 2008

ACTA CYBERNETICA

Information for authors. Acta Cybernetica publishes only original papers in the field of Computer Science. Manuscripts must be written in good English. Contributions are accepted for review with the understanding that the same work has not been published elsewhere. Papers previously published in conference proceedings, digests, preprints are eligible for consideration provided that the author informs the Editor at the time of submission and that the papers have undergone substantial revision. If authors have used their own previously published material as a basis for a new submission, they are required to cite the previous work(s) and very clearly indicate how the new submission offers substantively novel or different contributions beyond those of the previously published work(s). Each submission is peer-reviewed by at least two referees. The length of the review process depends on many factors such as the availability of an Editor and the time it takes to locate qualified reviewers. Usually, a review process takes 6 months to be completed. There are no page charges. Fifty reprints are supplied for each article published.

Manuscript Formatting Requirements. All submissions must include a title page with the following elements:

- title of the paper
- author name(s) and affiliation
- name, address and email of the corresponding author
- An abstract clearly stating the nature and significance of the paper. Abstracts must not include mathematical expressions or bibliographic references.

References should appear in a separate bibliography at the end of the paper, with items in alphabetical order referred to by numerals in square brackets. Please prepare your submission as one single PostScript or PDF file including all elements of the manuscript (title page, main text, illustrations, bibliography, etc.). Manuscripts must be submitted by email as a single attachment to either the most competent Editor, the Managing Editor, or the Editor-in-Chief. In addition, your email has to contain the information appearing on the title page as plain ASCII text. When your paper is accepted for publication, you will be asked to send the complete electronic version of your manuscript to the Managing Editor. For technical reasons we can only accept files in LaTeX format.

Subscription Information. Acta Cybernetica is published by the Institute of Informatics, University of Szeged, Hungary. Each volume consists of four issues, two issues are published in a calendar year. Subscription rates for one issue are as follows: 5000 Ft within Hungary, €40 outside Hungary. Special rates for distributors and bulk orders are available upon request from the publisher. Printed issues are delivered by surface mail in Europe, and by air mail to overseas countries. Claims for missing issues are accepted within six months from the publication date. Please address all requests to:

Acta Cybernetica, Institute of Informatics, University of Szeged
P.O. Box 652, H-6701 Szeged, Hungary
Tel: +36 62 546 396, Fax: +36 62 546 397, Email: acta@inf.u-szeged.hu

Web access. The above informations along with the contents of past issues are available at the Acta Cybernetica homepage http://www.inf.u-szeged.hu/actacybernetica/ .

EDITORIAL BOARD

Bálint Dömölki
IQSOFT
Budapest, Hungary
domolki@iqsoft.hu

Zoltán Ésik
Department of Foundations of
Computer Science
University of Szeged
Szeged, Hungary
ze@inf.u-szeged.hu

Zoltán Fülöp
Department of Foundations of
Computer Science
University of Szeged
Szeged, Hungary
fulop@inf.u-szeged.hu

Ferenc Gécseg
Department of Computer Algorithms
and Artificial Intelligence
University of Szeged
Szeged, Hungary
gecseg@inf.u-szeged.hu

Jozef Gruska
Institute of Informatics/Mathematics
Slovak Academy of Science
Bratislava, Slovakia
gruska@savba.sk

Tibor Gyimóthy
Department of Software Engineering
University of Szeged
Szeged, Hungary
gyimothy@inf.u-szeged.hu

Helmut Jürgensen
Department of Computer Science
Middlesex College
The University of Western Ontario
London, Canada
helmut@csd.uwo.ca

Alice Kelemenová
Institute of Computer Science
Silesian University at Opava
Opava, Czech Republic
Alica.Kelemenova@fpf.slu.cz

László Lovász
Department of Computer Science
Eötvös Loránd University
Budapest, Hungary
lovasz@cs.elte.hu

Gheorghe Păun
Institute of Mathematics of the
Romanian Academy
Bucharest, Romania
George.Paun@imar.ro

András Prékopa
Department of Operations Research
Eötvös Loránd University
Budapest, Hungary
prekopa@cs.elte.hu

Arto Salomaa
Department of Mathematics
University of Turku
Turku, Finland
asalomaa@utu.fi

László Varga
Department of Software Technology
and Methodology
Eötvös Loránd University
Budapest, Hungary
varga@ludens.elte.hu

Heiko Vogler
Department of Computer Science
Dresden University of Technology
Dresden, Germany
vogler@inf.tu-dresden.de

Gerhard J. Woeginger
Department of Mathematics and
Computer Science
Eindhoven University of Technology
Eindhoven, The Netherlands
gwoegi@win.tue.nl

INTELLIGENT SYSTEMS 2007 SECOND SYMPOSIUM OF YOUNG SCIENTISTS

Selected Papers

Guest Editors:

Tibor Gregorics

Department of Software Technology and Methodology
Eötvös Loránd University
Budapest, Hungary
gregorics@inf.elte.hu

Péter Szeredi

Department of Computer Science and Information Theory
Budapest University of Technology and Economics
Budapest, Hungary
szeredi@cs.bme.hu

Bálint Molnár

Department of Information Systems
Corvinus University of Budapest
Budapest, Hungary
molnar@informatika.bke.hu

Zoltán Vámossy

Institute of Software Technology
Budapest Tech
Budapest, Hungary
vamossy.zoltan@nik.bmf.hu

Edit Sántáné-Tóth

Department of Software Technology and Methodology
Eötvös Loránd University
Budapest, Hungary
santa@inf.elte.hu

László Zsolt Varga

System Development Department
Computer and Automation Research Institute of the Hungarian Academy of Sciences (MTA SZTAKI)
Budapest, Hungary
laszlo.varga@sztaki.hu

Preface

This issue of *Acta Cybernetica* contains eight papers originally presented at the Second Symposium of Young Scientists, entitled *Intelligent Systems 2007*, held in Budapest, on November 23, 2007. The Symposium, with the Hungarian acronym IRFIX'07, was organised by the Artificial Intelligence Section of the John von Neumann Computer Society (JvNCS), the Hungarian member of the European Coordinating Committee for Artificial Intelligence (ECCAI). The Programme Committee was led by the Chair of the AI Section of JvNCS, Péter Szeredi, and included the five members of the Executive Board of the AI Section, listed as Guest Editors below.

The main goal of the meeting was to provide a forum for young researchers in both theoretical and practical AI for presenting their work, and to support the exchange of ideas between the Hungarian research groups in AI. The Symposium was part of the Hungarian Science Festival, a month-long series of lectures, conferences, celebrations, and other events commemorating the founding of the Hungarian Academy of Sciences in 1825.

The IRFIX'07 Symposium included 12 talks and 12 poster presentations, in various subfields of Artificial Intelligence. The talks were organised in three sessions, entitled Declarative Technologies, Machine Learning, and Applications. The conference featured an invited talk by prof. Tibor Vámos, ECCAI fellow, entitled "What is the use of epistemology, the critical examination of knowledge?". There were over 50 participants, representing a broad range of Hungarian higher education and research institutions, as well as company research labs. There was a lively discussion after each talk, which continued during the concluding poster session.

The event was hosted by the John von Neumann Faculty of Informatics of the Budapest Tech, in its new building at Bécsi út, Budapest.

Following the Symposium the authors of both standard and poster presentations were invited to submit papers to this Special Issue of *Acta Cybernetica*. Fourteen papers were received and were then subjected to the normal refereeing process of the Journal. The eight accepted papers cover a broad spectrum of topics, and report on progress both in the theory and in the practice of AI. The first three papers discuss declarative technologies and their applications: a web application development framework (Hunyadi), a semantic approach to content management (Kovács), and Prolog-based tools for teaching chemistry (Pántya and Zsakó). The next three articles, all from the research group of András Lőrincz, deal with various aspects of reinforcement learning (Szita and Lőrincz, Jeni et al., Gyenes et al.). The Special Issue is concluded with two papers on AI algorithms and theory: a novel evolutionary algorithm is presented by Lomonosov and Renner, while Kárász discusses a mathematical model of a discrete retrial system.

Thanks are due to all the authors presenting their work at IRFIX'07 and especially those submitting their contributions to this Special Issue. In addition to the Guest Editors, the following colleagues took part in the reviewing process: Tibor Ásványi, Balázs Csanád Csaji, László Csink, Zoltán Istenes, Attila Kiss, László Lakatos, Gergely Lukácsy, József Kázmér Tar, and Zsolt Zombori. Their help is very much appreciated. Special thanks are due to Zoltán Vámossy, for his excellent work on the local organisation of the event.

The Third Symposium of Young Scientists, *Intelligent Systems 2008*, is held on November 28, 2008, and a similar Special Issue of Acta Cybernetica is scheduled for 2009.

Tibor Gregorics
Bálint Molnár
Edit Sántáné-Tóth

Péter Szeredi
Zoltán Vámossy
László Zsolt Varga

Guest Editors
Members of the Executive Board of the
Artificial Intelligence Section of JvNCS

Acta Cybernetica 18 (2008) 561–578.

Prosper: Developing Web Applications Strongly Integrated with Prolog

Levente Hunyadi*

Abstract

Separating presentation and application logic, defining presentation in a declarative way and automating recurring tasks are fundamental issues in rapid web application development. Albeit Prolog is widely employed in intelligent systems and knowledge discovery, creating a web interface for Prolog has been a cumbersome task producing poorly maintainable code, which hinders harnessing the power of Prolog in information systems. This paper presents a framework called Prosper that facilitates developing new or extending existing Prolog applications with a presentation front-end. The framework relies on Prolog to the greatest possible extent, supports code re-use, and integrates easily with web servers. As a result, Prosper simplifies the creation of complex, maintainable web applications running either independently or as part of a heterogeneous system without leaving the Prolog domain.

Keywords: Prolog, web application development framework, application integration, XML, heterogeneous systems

1 Introduction

In developing information systems, modelling complex business processes is a challenging task. The model has to cater for many exceptions to a general rule, has to adapt to current business demands and has to be able to cope with large volumes of heterogeneous data sources. Flexibility and quick development are key issues. On the other hand, Prolog can ease development in a variety of ways. By straightforward formalisation of business rules, it yields verifiable code that is still close to the application domain. Extended BNF-style grammars contribute to the flexibility of data transformations between different data pools. In addition, constraint programming and intelligent reasoning based on background knowledge are other fields where compact Prolog programs can be formulated for complex problems. In other words, the expressiveness of Prolog can contribute greatly to the development of information systems.

*Budapest University of Technology and Economics, Department of Automation and Applied Informatics, H-1111 Budapest, Goldmann György tér 3., Hungary. E-mail: hunyadi@aut.bme.hu

Using Prolog in information systems inevitably requires integration with other parts of a larger system. However, the execution model of Prolog and imperative languages differs substantially, making it difficult to embed Prolog code in a program written in C#, Java or C++. Even though libraries [16, 14] are available to support integration to some degree, the resultant code is often very obscure, type safety is not enforced and debugging code is problematic.

One solution to this problem is presentation-level integration, indicated by the proliferation of XML and web services. In this approach, it is not the applications themselves that are integrated but output produced by one application is consumed by the other, allowing the use of completely different programming languages in the process.

An interesting application field of presentation-level integration is component-based development of web portals. In this case, portals are not built as monolithic applications but are composed of small, fairly independent components, called web parts or portlets. Each component generates presentation-level code (e.g. XHTML or XML), which is combined into a whole by a web portal framework. In this scenario, Prolog can be used to generate parts of the portal that exhibit intelligent behaviour while the rest can be developed by means of conventional imperative programming languages.

The idea of generating HTML or XML output in Prolog is not new: several frameworks [5, 14, 10, 11] exist that give excellent support for structured display of data in web pages. However, neither of them promotes a clear definition of presentation (i.e. how data are displayed) that is distinct from application logic (i.e. the main job of the application). As a result, presentation and application (or business) logic are interleaved, which in most cases eventually leads to poor maintenance. In addition, complex presentation logic, such displaying parts of a page based on a condition, or displaying variations of a given content repetitively for each element of a list are tasks that cannot be accomplished in a generic manner. Moreover, it would be desirable that these tasks be purely restricted to authoring XHTML or XML documents, possibly by using special annotation.

The proposed system, named *PROlog Server Pages Extensible aRchitecture* (or *Prosper* in short) [8], aims to combine the advantages of conventional web application development methods (such as separation of presentation and application logic and declarative definition of presentation by means of XML) with the potential in the Prolog language in order to create more intelligent web portals. It supports integrating Prolog applications in existing information systems as well as extending existing Prolog applications with a web interface.

Prosper is implemented mainly in SWI-Prolog and partially in C. SWI-Prolog is compliant to part one of the Prolog ISO standard and has comprehensive support for multi-threading. ISO-compliance caters for portability while multi-threading helps harness the potential in parallel execution. Network communication interfaces have been written in C to ensure maximum performance. The Prosper project (including full source code) is available at SourceForge.net [13].

The rest of the paper is structured as follows. Section 2 gives a brief introduction to methodologies and technologies the proposed framework makes use of.

Section 3 elaborates on design trade-offs, inspects related work and analyses possible approaches to create a framework for developing a web front-end with special attention to the chosen approach. In Section 4, the architecture of the proposed framework is laid out. Communication of remote parties over a network can be broken down into a series of requests and associated replies: Section 5 traces the way a request produces a reply in Prosper by means of an example. Section 6 gives some implementation details and performance metrics while Section 7, with which this paper concludes, summarises contributions and outlines possible ways of extension and future work.

Throughout the paper, knowledge of Prolog with basics on SGML-languages (especially XML [3] and (X)HTML [12]) and some experience in developing web applications with an application development framework (such as ASP.NET [9] and/or Java [2]) is assumed. One should consult the indicated sources for further details on these technologies.

2 Background

Essentially, the web operates in a request-and-reply manner according to the HyperText Transfer Protocol [6]. First, the client formulates a *request* querying a document. The request is received by the server, which looks up the requested document in its file system and returns it to the client in *reply*. In the case of *dynamic content generation*, the request received by the server does not correspond to a file system entry but is forwarded to a possibly external application that outputs the reply based on the *request context* (request parameters, session information, user preferences, etc.). Web application development frameworks are inserted into the chain either in place of the server (e.g. Java web solutions) or between the server and the external application (e.g. the ASP.NET framework), and expose a programmer-friendly view of the web environment to the application developer.

Web frameworks taking the place of the server require a thorough implementation to provide general web service functionality (e.g. include serving static content) with sufficient security. For this end, it is often desirable to use a trusted web server behind which applications are placed rather than using a separate endpoint for each application. In this scenario, frameworks are often connected to servers by means of server APIs (application programming interfaces). Here, the application is loaded as a module of the server and the server forwards requests that match some criteria (e.g. URL pattern or extension) to the application instead of processing them itself. This is called *strong coupling*.

Common Gateway Interface (CGI) describes a protocol that provides *loose coupling*. In order to process matching requests, the server invokes an external application (i.e. one that is not integrated with the server) with the given request context (query parameters, user settings, etc.) and returns the output it produces to the client. Loose coupling separates the operating system processes involved, which therefore minimally affect each other, increasing flexibility. In addition, fatal errors in the application do not endanger the server. Nonetheless, repetitive invocation of

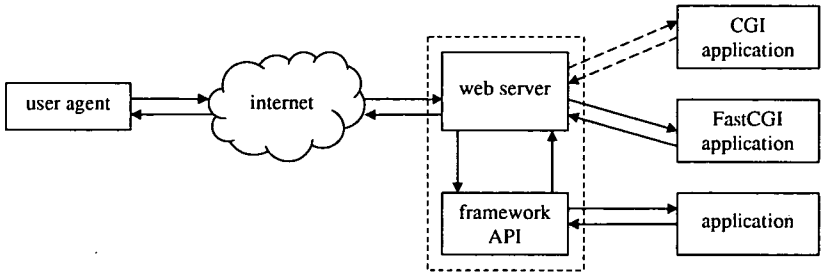


Figure 1: The web service chain. The dashed rectangle indicates the boundary of process space, continuous arrows refer to persistent, while dashed arrows to non-persistent connections.

an external program can take up valuable resources (by successive process initialisations, re-opening database connections, etc.). FastCGI [4] is a persistent version of the CGI protocol that allows applications to remain active after serving a request (thereby maintaining database connections, etc.) yet preserving their independence (i.e. no modification of server internals is required and the application works with web servers of multiple vendors).

The request and response chains and the relationships of the various application types are shown in Figure 1.

Applications that process many simultaneous requests have to be *multi-threaded* so that processing a request does not keep others waiting. Hence, each request is assigned a separate newly initialised thread. However, on high demand this can lead to the so-called *thrashing*, where threads with already assigned jobs lose computing resources to dozens of rapidly launched new threads, eventually leading to no thread performing useful task. Thus, applications often make use of the *worker thread model*. In this model, a constant number of threads execute concurrently. Jobs are assigned threads from a pool, which return to the pool after the job is complete. This allows fast processing of simultaneous requests with the elimination of thread startup costs and stability upon high demand.

Many web development platforms make use of the *model-view paradigm*. In this paradigm, *application logic* (what the program does) and visual *presentation* (how the results are displayed) are strictly separated. Presentation is defined in a declarative manner, often by means of a markup document (such as XML or XHTML), albeit additional code that drives the presentation layer (such as data binding or events) may be required in a *code-behind* file. On the other hand, application logic is written in the native language of the platform. While presentation may reference objects in application logic, the reverse is not true. This allows presentation and application logic to be created (more) independently and caters for easier maintenance of both. While not every web development framework makes it compulsory, the pattern can be considered fairly wide-spread, and is recognised as a key condition to creating complex web applications.

```
env(html, [], [
  env(body, [lang=hu],
    heading(1, title),
    text,
    ref('http://www.aut.bme.hu', hyperlink)
  ])
])
```

```
<html>
  <body lang="hu">
    <h1>title</h1>
    text
    <a href="http://www.aut.bme.hu">hyperlink</a>
  </body>
</html>
```

Figure 2: A PiLLoW Prolog term (above) and the equivalent HTML document it produces (below, ignoring white space).

3 Possible approaches and related work

In order to expose a web front-end, an application has to emit XML or (X)HTML documents based on application logic. This can be accomplished in two different ways:

1. producing and emitting presentation markup code directly using the platform language, or
2. embedding snippets of code written in the platform language within presentation markup code.

Generating web content directly in a Prolog program (1st approach), possibly with the help of general-purpose libraries, is fairly straightforward. The PiLLoW library [5], available in many Prolog implementations, is a notable representative of this approach. As exemplified by the library, the close relationship of Prolog terms and the hierarchical structure of HTML easily lends itself to composing the web page in the form of terms (Figure 2), which are then transformed to and output as plain text on demand. By means of uninstantiated variables in the term representation, simple templates can be created.

Nevertheless, a Prolog term representation is inherently not visual and integrates poorly into existing web authoring tools (such as web page designers). Moreover, the approach does not promote clear separation of application logic and presentation, so that programmers are tempted to violate the model-view paradigm, which eventually leads to more difficult maintenance. Also, a stand-alone Prolog server replying to requests on a dedicated port is often assumed, which is hard to

```
<?, member(number=N, Get),
    forall( ( between(1, N, X), factorial(X, Y) ),
?>
            <li>The factorial of <?= X ?> is <?= Y ?>.</li>
<?  ), ?>
```

Figure 3: An excerpt from a dynamically generated HTML server page composed with embedded Prolog escape sequences. The snippet lists all factorials from 1 to *N*. *N* is specified as a query string parameter.

incorporate into a complex environment with an existing web server. However, a library such as PiLLoW can relieve the programmer from the majority of recurring tasks and can thus contribute greatly to web application development, especially in simple scenarios. Commonly aided tasks include parsing HTTP GET and POST parameters, generating forms, HTTP cookies and session maintenance.

Embedding pieces of Prolog in the presentation layer (2nd approach) is another natural approach, which can be thought of as the "inside out" version of the previous one, motivated by various successful server-side technologies such as PHP [1]. Here, web pages are composed as (X)HTML rather than as Prolog terms, and Prolog calls are inserted in the text by means of special escape sequences. The helper library parses the page into a predicate consisting of a series of *write/1* statements and the equivalents of the embedded Prolog calls. Many projects that take this approach exist in the Prolog domain, [10] and [11] are two such examples.

Albeit simple, this approach is generally insufficient for larger projects as it is weakly structured. Apparently, even repetitively displaying a variation of a block of text as in Figure 3 produces code that is difficult to comprehend. More complex nesting is even harder to implement merely by means of skipping in and out of escaped blocks. Clearly, escape sequences lead to interleaved application logic and presentation, and are hence extremely difficult to maintain or extend.

Another variant of the second approach is composing web pages in an external framework, such as JSP or ASP.NET, and embedding foreign language calls to Prolog. PrologBeans for Java and PrologBeans.NET for the .NET platform [14], both available as SICStus extensions, are representatives of this variant. Here, all web-related issues are handled by an external framework, which provides optimised solutions to general patterns in web authoring and offers rapid application development. In order to call Prolog predicates, however, wrapper objects, written in the native language of the framework, are required that marshal calls to the Prolog engine. In fact, from a design perspective, the approach entails two parts, so-called *stubs*. The wrapper object constitutes the first stub, while its Prolog counterpart the other. The stubs maintain a TCP or piped connection to each other through which Prolog call parameters and results are transmitted, usually as a stream of characters.

While practical in harnessing the benefits of a web development framework, this approach undoubtedly requires experience in programming both Prolog and the

```
<html logic-module="factorial">
  <h1>Factorial example</h1>
  <psp:assign var="E" expr="{atom_number(http_get(number))}">
    <psp:for-all function="between(1, E)" iterator="N">
      <li><psp:insert function="factorial(N)" /></li>
    </psp:for-all>
  </psp:assign>
</html>
```

```
:- module(factorial, [factorial/2]).
factorial(Number, Factorial) :- ...
```

Figure 4: The Prosper example document *factorial.xhtml* (above) and the Prolog module *factorial.pl* associated with it (below). Some XHTML elements (e.g. *ul*) have been omitted and full namespaces are not shown for brevity.

external encapsulating language. From a performance point of view, stubs introduce a further level of indirection into the web service chain and often operate inefficiently because the Prolog and the foreign language execution model are vastly different. Lastly, debugging Prolog embedded in foreign code is substantially harder, which can greatly increase development time.

Prosper offers a balanced mix of the two main approaches. It is a variant of the first approach in the sense that the majority of request processing and content generation is performed in Prolog or Prolog-integrated libraries. Only Prolog programming experience is required and development is eased through improved debugging. On the other hand, it is closer to the second approach in the sense that it adopts the model-view paradigm of rapid application development frameworks by splitting web applications into an application logic and a presentation layer. Application logic is coded as a regular Prolog module, while presentation is an (X)HTML document with some elements and attributes carrying extra information for Prosper to realise so-called visual transformation rules (to be explained in detail). Figure 4 shows a web page and the associated application logic that lists all factorials up to N, functionally equivalent to the web page generated by the snippet in Figure 3. Despite its verbosity, the presentation layer is not interleaved with application logic and retains its structure as a regular XHTML document. Roughly speaking, Prosper can be viewed as an extension of PiLLoW with a more robust visual front-end. Section 4 elaborates on the design of the proposed framework.

4 Architectural overview

From a design perspective, Prosper can be decomposed into two major layers (Figure 5). The lower layer, *Prolog Web Container*, either acts as a stand-alone web server or maintains a direct persistent connection to the web server through the

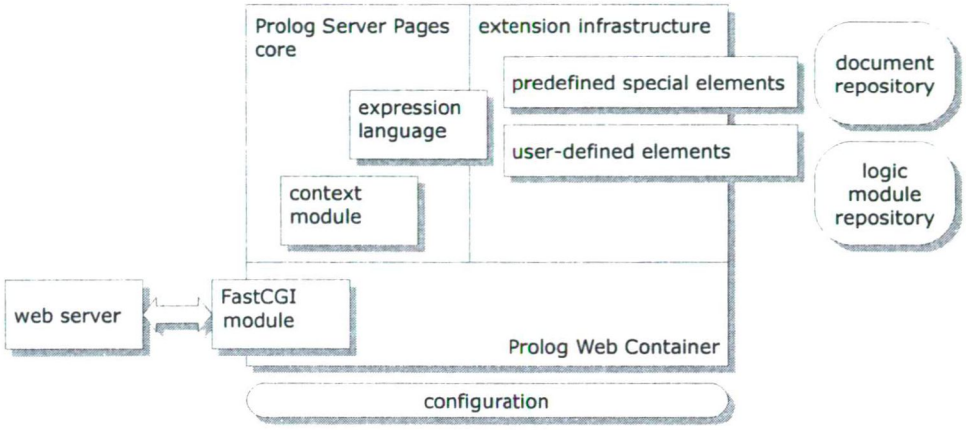


Figure 5: The architecture of the proposed framework.

FastCGI protocol. The FastCGI module transmits data to and from the Prolog framework. In addition to acting as or communicating with the web server, Prolog Web Container parses headers and payload associated with HTTP requests into Prolog terms and generates them for replies, maintains a worker thread pool and assigns jobs to threads. The primary task of the container is to isolate the communication protocol and provide a natural view of request and session data for the programmer. In accordance, the container provides similar facilities as other Prolog libraries in use, PiLLoW in particular, i.e. reversing content encoding, parsing query strings, etc.

Prolog Server Pages, built on top of the container, defines an XML-based document model. The conventional XML document model is extended with *special elements* belonging to a dedicated namespace each of which realises a *transformation rule*. A transformation rule describes how the (visual) content of an element is transformed based on attributes, and the local and global *context* of the given element. Local context corresponds to variables instantiated in server documents, while global context refers to request context as extracted by Prolog Web Container and exposed as Prolog predicates by the *context assertion* module. In assigning values to local variables, Prosper offers the so-called *expression language*. Expression language can be seen as an extension to the *is/2* predicate to include basic atom manipulation, request context variables and user-defined functions.[1]

Prosper includes a predefined set of special elements implementing the most common transformation rules such as conditionals and iteration constructs. However, the set of transformation rules is not restricted. Relying on the *extension*

[1] In this paper, a Prolog *function* corresponds to a predicate all of whose arguments are strictly inbound except for the last, which is strictly outbound, and which should be unified with a ground term and is interpreted as the return value of the function. This corresponds to the Mercury [7] definition of function.

infrastructure, the user may create new modules that contain hook predicates registered for steps associated with reply generation. Modules correspond to XML namespaces and exported hook predicate names to element names in server page documents. In fact, it is via hook predicates that the predefined transformation rules are realised in the framework, which means – in the extreme case – that they can also be redefined. Special elements and their implementor hook predicates are declared in a *configuration file*. The configuration file also holds connection settings to the web server and parallel execution parameters required by Prolog Web Container.

Apart from the visual part of Prolog Server Pages, the *logic modules* give real power to the architecture. While independent from Prolog Server Pages documents, they provide the code-behind that encapsulates true application logic as conventional Prolog modules. Server pages can reference code-behind in a variety of ways: assign server page variables based on application logic, test for the satisfiability of predicates (goals), and formulate conditions using the return value of functions, each of which may affect visual layout.

Prolog modules constituting application code reside in a dedicated directory, the so-called *logic module repository*. Similarly, Prosper maintains a *document repository*, which is the default location to search for server pages.

5 Generating a reply

In order to get a deeper insight into the internals of the framework, in this section we will trace how a request dynamically produces a reply in Prosper. As an example, let us suppose that the user has entered a URL into his browser's location bar that corresponds to a web page which lists all factorials up to 3 (e.g. *http://prosper.cs.bme.hu/factorial.psp?number=3*). Albeit the example is simple, it will illustrate the different stages of the service chain.

Once received by the web server, based on configuration settings, the server detects that this HTTP request is to be forwarded to Prosper for reply generation. It dispatches a FastCGI request, which is intercepted by one of the idle Prolog Web Container worker threads.[2] The thread extracts the context associated with the request as Prolog terms. The context typically includes query parameters in the URL (typically for HTTP GET requests), HTML form data passed as payload (typically for HTTP POST requests) and the session identifier. The Prolog representation of the context is handed over to Prolog Server Pages. In our example, the request context only contains GET parameters, represented by the list *[number='3']*.

First, Prolog Server Pages *loads* the document associated with the URL. The loaded document is *preprocessed* into a so-called intermediate term (IT) representation. Context is then *asserted* into a dedicated module and the document is *evaluated*. Evaluation ends with generating *output*, which is returned by Prolog Web Container to the web server as a FastCGI reply (Figure 6). So-called *transformation rules* are associated with both the preprocessing and the evaluation phase.

[2] See predicate *worker/1* in module *prosper_server* [13].

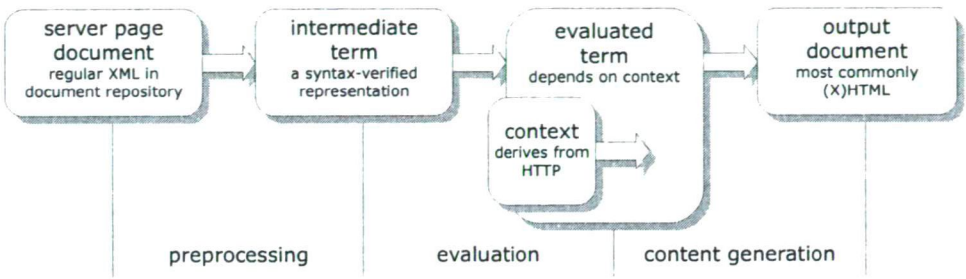


Figure 6: The phases of reply generation for server page documents that have not been cached.

The crucial difference is that in the preprocessing phase, no external context information is available, while evaluation-time transformation is context-sensitive. The aforementioned steps are elaborated below.

Loading a Prosper document. The role of the loading phase[3] is to fetch a referenced document from disk and construct its Prolog XML term representation, similar to the one used by the PiLLoW library [5].

Whenever an HTTP request corresponds to a server page that has not been loaded, Prosper looks for the page in the document repository. Let us suppose that the URL entered by the imaginary user does not correspond to a loaded document. Therefore, the document *factorial.xhtml* (Figure 4) is loaded and parsed into a Prolog XML term representation as seen in Figure 7. This representation mainly consists of nested *element/3* terms, where the arguments represent:

1. the name of the XML element after namespace resolution;
2. a list of attributes associated with the element;
3. a list of nested XML nodes as *element/3* terms for XML elements or atoms for character data.

Preprocessing phase. The goal of the preprocessing phase,[4] the next link in the service chain, is to validate the loaded document. Preprocessing ensures that special elements referenced by the document exist, they are used correctly in terms of syntax, and that the logic module associated with the document is loaded and compiled.

As previously mentioned, special elements correspond to transformation rules. What the transformation rule exactly does depends on the attributes associated with the element and its context. In our example, *psp:assign*, *psp:for-all* and

[3]Implemented in *import_page/3* in module *prosper_core* [13].
[4]Implemented in *markup_to_term/6* in module *prosper_core* [13].

```
elem(html, ['logic-module'=factorial], [
  elem(h1, [ ], ['Factorial example']),
  elem(psp:assign, [var='E', expr='{atom_number(http_get(number))}'], [
    elem(psp:for-all, [function='between(1, E)', iterator='N'], [
      elem(li, [ ], [
        elem(psp:insert, [function='factorial(N)'], [ ])
      ])
    ])
  ])
])
```

Figure 7: The XML term representation of the example document in Figure 4.

psp:insert are special elements, assuming the namespace *psp* is registered with Prosper.[5] The *psp:assign* special element can have a *var* attribute, which specifies the name of the variable to introduce in the scope of the element. Similarly, *psp:insert* is used with the attribute *function* in the example to insert a return value but could also be used in conjunction with *expr* to insert the value of an expression.

However, in the preprocessing phase no context information associated with the HTTP request is available; it has not yet been asserted. In spite of this, verifying attributes, parsing atoms into terms, etc. are already possible. These operations are performed by *preprocessing-time hooks* for each special element. A hook predicate interprets element attributes and/or contents and has the following signature (*elementName* denotes the name of the special element without the namespace):

elementName(+VarTypes, +Attrs, +Contents, -Terms)

Here, *VarTypes* propagates type information,[6] *Attrs* is a list of *Name=Value* pairs, which consists of attributes that parametrise the element. *Contents* is a list of inner elements in XML term representation. *Terms* is the single output argument of the predicate, which is the IT representation (preprocessed form) of the element and is commonly bound to a single-element list of the following form:[7]

[extension(ModuleName:Predicate, ContentTerms)].

In this term, *ContentTerms* has similar semantics as *Terms* in the enclosing element: it is the IT representation of the enclosed child elements. This suggests a recursive way of operation. Indeed, albeit not compulsory, most special elements compute their own IT representation based on that of their descendants. *Predicate* corresponds to a Prolog predicate, which (augmented with some additional arguments) will be called in the evaluation phase to generate output. In other words,

[5] For conciseness, namespaces are not written out as full URLs, even though in the actual implementation, they are used in that manner.

[6] Although there is no strict typing scheme in Prolog, some degree of type enforcement allows catching errors at an earlier stage.

[7] In fact, *Terms* is a list of atoms, *element/3* and *extension/2* terms. However, only *extension/2* terms are subject to evaluation in a later phase thus *Terms* is usually a list with a single *extension/2* element.

```
elem(html, [ ], [
  elem(h1, [ ], ['Factorial example']),
  extension(assign_expression('E', EL), [
    extension(for_all(factorial:between(1, E)-['E'=E], 'N'), [
      elem(li, [ ], [
        extension(insert_function(factorial:factorial(N)-['N'=N]), [ ])
      ])
    ])
  ])
])
```

Figure 8: The intermediate term representation of the example document. *EL* denotes the execution plan of the expression language term and is omitted for conciseness.

Predicate is the evaluation-time transformation rule associated with the special element parametrised with *Attrs*. For instance, a different *Predicate* is associated with a *psp:assign* element if it assigns a variable based on an expression than if based on a function call. In fact, arguments present in *Attrs* as an association list are converted into positional arguments with appropriate conversions where necessary (e.g. atoms converted to Prolog goals).

Hook predicates should never fail but should signal malformed syntax (such as an unrecognised attribute) by throwing an exception.[8] This guarantees that the document is syntactically well-formed at the end of the preprocessing phase.

In order to better comprehend the preprocessing phase, we compare the XML term representation of our example document in Figure 7 with its preprocessed version in Figure 8.

In the case of the root element *html*, the two representations are identical, except for the attribute *logic-module*. This attribute binds a conventional Prolog module (application logic or presentation code-behind) to the Prosper document. The exact location of the module source file is either directly specified in the *logic-module* attribute as an absolute path, or it may be a relative path, in which case it is searched for w.r.t. the module repository. Any predicates that occur in the document are auto-qualified with the name of this module during the preprocessing phase.

The first notable difference is *psp:assign*, which has been converted into an *extension/2* term. *assign_expression* is the name of a predicate that computes an expression language (EL) term and assigns its value to a (server page) variable. The scope of the variable is the enclosed contents of the *psp:assign* element. The function *http_get* in the EL term returns the string value of a query string variable, while *atom_number*, as its name suggests, converts its operand to a Prolog number. Just as documents, EL expressions are preprocessed, yielding an *execution plan*,

[8]The rationale behind throwing an exception is that simple failure prevents extracting the context of the actual error, which is necessary in order to print the proper error message.

which is not shown in Figure 8. The execution plan is a compound term that contains

1. the uninstantiated variables in the expression, and
2. the module-qualified names of the Prolog functions to call to compute the result.

The representation of the special element *psp:for-all* has also changed substantially. The atom in its attribute called *function* has been converted into a real Prolog term augmented with a list of uninstantiated variables in it. *for_all(Function-Insts, Variable)* is a predicate that instantiates variables in *Function* and calls it, returning results in a local variable. Subsequent solutions are obtained through backtracking. Note the number of arguments to *between/3* (the third, output argument is absent and is appended automatically) and the auto-qualification.

For the sake of higher performance, Prosper caches preprocessed documents. If a document is available in the cache, the loading and preprocessing phases are skipped.

Request context assertion. Context information available in Prosper documents and logic modules is loaded in the request context assertion phase. The primary goal of this phase is to expose HTTP request context (such as request parameters and session variables) to EL functions and logic module predicates in a natural manner without having to propagate an extra argument encapsulating the context. Predicates in the module *psp* store context information by means of *thread-local* blackboard primitives [15]. Whenever a mutable (session) value is modified while the request is served (e.g. a session variable is assigned to), changes are recorded in a (thread-global) dynamic fact database at the end of the subsequent evaluation phase. Hence, no particular thread is associated with any session and any worker thread may serve any request. Worker threads load current values from the dynamic fact database into blackboard primitives before the evaluation phase and record new values when evaluation ends.

Logic modules have access to context by calling predicates exported by the module *psp*. For instance, the *http_get/2* and *session/2* predicates retrieve the value of a GET and a session variable, respectively, and the predicate *session_set/2* assigns a value to a session variable.[9] For maximum conformance to the Prolog execution model, they all support backtracking, i.e. assignments to session variables are undone upon failure in a logic module predicate.[10]

Evaluation phase. In the last major phase, *evaluation*,[11] the preprocessed document is transformed w.r.t. the available request context. By the end of the evaluation phase, the document has been transformed into a term representation whose string equivalent is ready to be sent back directly to the client as response.

[9] For a full list, see exported predicates in module *psp* [13].

[10] SWI-Prolog provides backtrackable destructive assignment on blackboard primitives.

[11] Implemented in *term_to_elements/3* in module *prosper_extensions* [13].

From a declarative point of view, each IT element represents an (evaluation-time) transformation rule, influenced by

1. term contents,
2. asserted HTTP request- and session-related data that is globally accessible in the entire document, and
3. local variables assigned by outer special elements (i.e. that encapsulate the element to which the rule corresponds) such as *psp:assign*.

In the case of *element/3* terms, the (recursive) transformation rule is trivial: transformation rules are applied to each child element with the same context as for the parent element and the evaluated form of the parent element comprises of the combined results of these transformation rules. For *extension/2* terms, recall that the first argument corresponds to a hook predicate assembled in the preprocessing phase: this is what represents the transformation rule. From a procedural point of view, in fact, the IT representation is traversed top-down, at each depth invoking hook predicates or the trivial transformation rule, where hook predicates may introduce new local variables before processing the children of the term they correspond to.

Local variables are means to store and reuse calculated data within server page documents. In contrast to globally available data (loaded into the thread-local module *psp* in the context assertion phase), they are accessed as Prolog variables rather than predicates and they are confined to the server page document in which they are introduced and may not be directly used in presentation code-behind or application logic files. More precisely, the scope of local variables is always restricted to the descendants of the element in which they are assigned and are hidden by variables of the same name. Server page local variables have similar semantics as Prolog or XSLT variables in the sense that they can be assigned only once. Contrary to Prolog, however, variables cannot remain uninstantiated and are unified immediately in the element in which they are introduced.

Figure 9 shows the evaluated version of the preprocessed document in Figure 8. The result should not be surprising. For the elements *html*, *h1* and *li*, the trivial transformation rule has been applied and they are intact except for their recursively processed contents. The IT equivalents of special elements *psp:assign*, *psp:for-all* and *psp:insert* are absent from the output but their effect is apparent. The local variable *E*, which is introduced by *assign_expression*, has been used to instantiate unbound variables in the function *between(1, E)*, and the iteration variable *N* of *for_all* has been used multiple times to call the function *factorial(N)*. *N* behaves as expected, taking a different value for each loop of the iteration.

From the perspective of the framework, local variables are in fact *Name=Value* members in an association list. The association list is initially empty for the root element but may be extended with further members by any transformation rule, in which case the recursively processed descendant elements see the extended list. In our example, the evaluation-time transformation rule associated with the *psp:assign*

```
elem(html, [ ], [
  elem(h1, [ ], ['Factorial example']),
    elem(li, [ ], ['1']),
    elem(li, [ ], ['2']),
    elem(li, [ ], ['6'])
  ])
])
```

Figure 9: The evaluated form of the example document.

element prepends the variable E to the name-value list, while the rule related to *psp:for-all* does so with N.

6 Implementation

Prosper is implemented mainly in SWI-Prolog and partially in C. The most notable SWI-specific extra services utilised by the framework are XML document parsing and generation, blackboard primitives, multi-threading and basic thread communication.

The framework comprises of the following major components:

1. The *server module* implements Prolog Web Container.
2. The *core module* manages the lifecycle of a Prosper page. In particular, it imports pages on demand, initiates context assertion, page preprocessing and evaluation, and outputs error documents.
3. The *context module* asserts and retracts thread-local data via blackboard primitives to expose request, session and user preference values, all of which are manipulated through dedicated predicates of the module *psp*.
4. The *extension module* contains predicates essential to special element implementors. It includes helper predicates to aid XML attribute parsing and the predicates *element_to_terms/3* and *term_to_elements/3*, which realise page preprocessing and evaluation, respectively. The latter two predicates are called by transformation rule hooks to recursively process child elements.
5. The *built-in elements module* contains the predefined set of special elements, including simple and compound conditionals, iteration constructs, variable assignment and insertion.
6. The *expression language module* is responsible for expression language execution plan generation and expression evaluation.
7. The *FastCGI foreign language module*, written in C, implements the FastCGI protocol.

Table 1: Comparative performance of various frameworks.

Development tool	Application model	small	large	intensive
SICStus Prolog 3.12.5	CGI, saved state	165.78	225.33	n.a.
SWI-Prolog 5.5.33	CGI, saved state	39.47	60.17	n.a.
PrologBeans.NET	ASPX	6.297	7.781	91.91
Prosper (PWC + PSP)	multi-threaded FCGI	2.688	8.719	5.828
Prosper (PWC only)	multi-threaded FCGI	1.938	6.953	n.a.
SWI-Prolog 5.6.27	standalone server	1.266	6.313	n.a.
static html content		0.875	1.406	n.a.

While primarily designed to increase designer and programmer effectiveness, the proposed architecture is comparable to other Prolog-based technologies in terms of speed. In a loopback scenario (i.e. server and client were running on the same machine), different configurations were polled by HTTP requests with GET parameters. All configurations parsed the query string, computed a simple arithmetic expression based on query parameters, and displayed results in a web page. CGI and FastCGI-based applications (Prosper inclusive) connected to Apache/2.0.54, .NET applications ran on the built-in web server provided with Visual Studio 2005. Benchmarking was performed by ApacheBench 2.0.41 on an AMD Athlon64 3000+ running Microsoft Windows XP Professional SP2.

Table 1 shows cumulative response times in seconds for 1000 requests with 2 concurrent threads. In test cases *small* and *large*, responses of about sizes 1kB and 50kB were requested with few embedded Prolog calls. In test case *intensive*, the architectures had to call about 50 Prolog predicates in application logic to produce a result of about 3kB in size. n.a. indicates that there is no overhead of a Prolog call-intensive setup (i.e. presentation and application logic are not separated and both are in Prolog) or it is not meaningful for the test case. Static HTML content is included for reference, and is meant to indicate the absolute lower bound for response time as (unlike the other scenarios) it requires no extra overhead owing to context evaluation.

Three cases are of special interest. The standalone multi-threaded HTTP server shipped with SWI-Prolog can serve as the basis for comparing the performance of Prolog-based frameworks. It provides convenience tools for HTTP reply generation but intermixes presentation and application logic. The difference in speed between Prosper with Prolog Web Container and SWI's standalone server gives an estimate of the cost of using an intermediary FastCGI transmission. The extra overhead of Prosper with the Prolog Server Pages document model shows the relative cost of having a separate presentation and application logic layer. The careful reader may also notice that Prosper is slower than PrologBeans.NET when large documents with few Prolog calls are served. This is attributable to the greater efficiency ASP.NET handles strings than Prolog handles atoms in a multi-threaded environment.

7 Summary, perspectives for future work

In this paper, a framework that facilitates developing web-oriented Prolog applications has been presented. With a persistent multi-threaded architecture, an XML-based document model and a set of reusable transformation rules, it provides an efficient yet convenient way to create web applications in Prolog. Code changes required in existing Prolog modules for the sake of web presentation are minimal and web pages constituting the presentation layer can be composed with a declarative way of thinking in any arbitrary XML editor. Presentation and application logic are clearly separated, thus application logic can be debugged and maintained independently. Lastly, the framework integrates well in existing web server scenarios and is open to extension.

As seen in the factorial example, the current implementation of Prosper generates content by traversing a term that constitutes some transformed version of the presentation source document. Compilation of these source documents into Prolog predicates could contribute to increased performance, especially in the case of long documents that have considerably large tree equivalents.

Besides dynamic content generation based on Prolog application code, web services offer another approach to integrate Prolog into complex information systems. As Prosper provides a straightforward way to parse and generate markup content, consuming and producing SOAP envelopes corresponding to Prolog calls and solutions seems a natural future extension.

Acknowledgements

The author acknowledges the support of the Hungarian NKFP Programme for the SINTAGMA project under grant no. 2/052/2004.

References

[1] Achour, Mehdi, Betz, Friedhelm, Dovgal, Antony, Lopes, Nuno, Richter, Georg, Seguy, Damien, Vrana, Jakub, et al. *PHP Manual.* PHP Documentation Group, 2007. `http://www.php.net/manual/en/`.

[2] Armstrong, Eric et al. *The J2EE 1.4 Tutorial (For Sun Java System Application Server Platform Edition 8.1 2005Q2 UR2).* Sun Microsystems, June 2005.

[3] Bray, Tim et al., editors. *Extensible Markup Language (XML) 1.0.* World Wide Web Consortium, 4th edition, August 2006. `http://www.w3.org/TR/2006/REC-xml-20060816/`.

[4] Brown, Mark R. *FastCGI specification.* Open Market, Inc., April 1996. Document Version: 1.0.

[5] Cabeza, Daniel and Hermenegildo, Manuel. *The PiLLoW Web Programming Library.* The CLIP Group, School of Computer Science, Technical University of Madrid, January 2001. `http://www.clip.dia.fi.upm.es/Software/pillow/pillow.html`.

[6] Fielding, R. et al. *Hypertext Transfer Protocol – HTTP/1.1.* Network Working Group, The Intenet Society, June 1999. RFC 2616.

[7] Henderson, Fergus et al. *The Mercury Language Reference Manual.* University of Melbourne, 2006. Version 0.12.2.

[8] Hunyadi, Levente. Prosper: A framework for extending Prolog applications with a web interface. In Dahl, Verónica and Niemelä, Ilkka, editors, *Proceedings of the 23rd International Conference on Logic Programming*, Logic Programming, pages 432–433, Porto, Portugal, September 2007. Springer. LNCS 4670.

[9] Hurwitz, Dan and Liberty, Jesse. *Programming ASP.NET.* O'Reilly, 3rd edition, October 2005.

[10] Johnston, Benjamin. Prolog server pages. `http://www.benjaminjohnston.com.au/template.prolog?t=psp`, 2007.

[11] Nuzzo, Mauro Di. Prolog Server Pages: A server-side scripting language based on Prolog. `http://www.prologonlinereference.org/psp.psp`, April 2006.

[12] Pemberton, Steven et al. *XHTML 1.0 The Extensible HyperText Markup Language.* World Wide Web Consortium, 2nd edition, August 2002.

[13] Prosper hosted at SourceForge.net. `http://prospear.sourceforge.net/`.

[14] Swedish Institute of Computer Science. *SICStus Prolog Users Manual*, May 2007. Release 4.0.1 `http://www.sics.se/sicstus/docs/latest/html/sicstus/PrologBeans.html`.

[15] Wielemaker, Jan. Native preemptive threads in SWI-Prolog. In Palamidessi, Catuscia, editor, *Practical Aspects of Declarative Languages*, pages 331–345, Berlin, Germany, December 2003. Springer Verlag. LNCS 2916.

[16] Wielemaker, Jan and Anjewierden, Anjo. An architecture for making object-oriented systems available from Prolog. In *WLPE*, pages 97–110, 2002.

Acta Cybernetica 18 (4) 579–593.

Improving Content Management – A Semantic Approach

Barna Kovács*

Abstract

Public administration institutions – as well as citizens and businesses – have to meet challenges of the constantly changing business and legal environment. The complexity and quantity of information to be faced with by these actors is increasing at an alarming rate. Research and development projects must turn to the development of innovative, modern technologies which enable citizens and businesses to access, understand and apply complex information easily. Ontology-based content management systems can contribute to the improvement of quality and effectiveness of significant processes, requiring the application of complex information, within the public administration or in a corporation. Compared to traditional content management systems, these systems can support further functions, such as semantic enabled search, explication of relations between documents, drafting of new documents, and version management, as well. Ontologies, in addition to the definition of concepts, support the most detailed and complete exploration of semantic relations between the concepts of a given domain.

Keywords: Ontology-based content management, Semantic-enabled content management, Content management systems, Ontologies, Semantic technologies

1 Introduction

Citizens, businesses, and even public administration institutions have to meet challenges provided by the constantly changing business and legal environment. The complexity and quantity of laws and regulations is increasing at an alarming rate. Consequently, research and development projects must turn to the development of innovative, modern technologies enabling citizens and businesses to access, understand and apply complex legislation and regulations easily. Ontology-based content management systems can contribute to the improvement of quality and efficiency of significant processes of public administration requiring the application of complex laws and other legal sources.

*Corvinus University of Budapest, 1093 Budapest, Fővám tér 8., bkovacs@informatika.uni-corvinus.hu

Content management systems (CMSs) support the creation, management, distribution, publication, and discovery of corporate information. This definition is strongly coupled to knowledge management and has close ties with the management and presentation of information. In contrast with common opinion, content management systems are more than just web content management systems, which are designed to build web and community portals.

Ontology-based content management systems can support further functions, such as semantic enabled search, explication of relations between documents, drafting of new documents, and version management of documents, as well. Ontologies, in addition to the definition of concepts, support the most detailed and complete exploration of semantic relations between the concepts of a given domain. Legal sources, for example, have been coming into existence in many forms and formats, from different jurisdictions, in various languages, with diverse internal structures. An ontology-based approach provides support for capturing patterns in a single framework which is general enough to create a representation of requirements for content management of legal rules codified in multiple jurisdictions. This approach also enables establishing links with legal knowledge systems and legal resources and provides help in maintaining the knowledge base of the CMS as the law is being changed.

2 Content Management

2.1 What is Content?

2.1.1 Digital Content

Providing a precise definition of content is a real challenge. In the literature, there exist numerous definitions for content; definitions differ from each other depending on the authors' focus area. According to Wordnet, the meaning of the word *content* varies from one context to another. Content is 'everything included in a collection'. In the context of messages, subject matter and substance are considered content, i.e. anything related to communication [19]. A definition more adequate to organisational life can be found in [10]: 'The material, including text and images, that constitutes a publication or a document.'

When content is considered in today's computerised organisations, usually digital content is assumed. In this sense, content can be defined as

> 'a commonly used term with regard to the Internet and other electronic media [...]. In its broader sense it refers to material which is of interest to users, such as textual information, images, music and movies, and it generally excludes (1) formatting information, [...], (2) software that is used to provide and render [...] it and (3) unrelated advertising' [16]

The distinction of digital content is necessary, since there are lots of other types

of content available in an organisation that are on paper or in other forms which cannot be easily handled by information systems – except when they are digitised.

Information systems deal with the production, processing and retrieving of digital content. The cost of producing digital content is very low compared to the production of any other type of content. This can lead to information overload very soon, which is a relevant problem of today's information sciences. Accordingly, decreasing costs of information creation increase the cost of information processing, requiring human work in most of the cases.

2.1.2 Textual Content

In a public administration environment, in the majority of cases, content takes the form of documents. Documents represent textual data in an unstructured manner, which makes their processing more difficult. Similarly, there are lots of textual data appearing in an organisation, which cannot be considered documents but can be equally important. Communication logs (email or discussion threads) are good examples of such textual data. Currently, other types of content (such as audio, video materials or pictures) can be considered less important in an organisation, except in cases where they are crucial resources.

Computer programs can process textual data, since this format can be indexed and searched easily. Processing of non-textual data, in contrast, is a white area on the map of information processing. Currently only humans can describe the content of certain data items, such as videos, audio materials or pictures. These descriptions can be stored as meta-data. However, producing such meta-data requires tremendous work and still does not ensure reliable results.

2.1.3 How does Content Differ from Regular Data?

As we have seen so far, content and data are related, moreover quite similar terms. Data itself does not have a certain meaning and usually is a broader term than content [15]. However, content itself can contain data, as well as information or knowledge[1], depending on the creator's intention. This way, similarly to data, meta-data can also be assigned to content [14]. Content usually has a specific context, name or title, and other kinds of meta-data associated with it, which can be specified using the Dublin Core standard [7], for example. As another example, consider the meta-data stored by a telecommunication company for every phone call.

2.2 Managing Content

2.2.1 Content Lifecycle

In the current context, content lifecycle and document lifecycle can be considered equal. Usually, four phases of content lifecycle are identified, although some au-

[1] We refer to the system theory approach to data, information and knowledge, as presented by [1]

thors use a different number of stages. Authors usually agree on the first phase (content creation) and on the publishing phase. In between these two there is an editorial process. Furthermore, a final stage can be added, when the document is retired [6]. Consider the following example: one or more authors create a document, which might go through an editorial process of subsequent updates, and then it gets published. In this published form it can also be updated several times before it is finally outdated or retired for some reasons, and becomes a subject of archiving or deletion. Content management systems should support these phases in a collaborative manner, since several people can work together on the same content.

Non-textual content can have the same lifecycle pattern. However, it might be more difficult to create or update the content due to the natural characteristics of non-textual data. Content management systems should support the whole lifecycle in this case as well.

2.2.2 What is a Content Management System?

The goal of content management is to help the users in creating, organising, or, in other words, managing data, information or knowledge represented in the form of content. This also includes the retrieval of information, which is one of the greatest challenges for information systems. Retrieving information means the ability of the system to find data which are relevant for a given user. If data is found to be relevant to the user's problem, then it can be considered as information relevant to the given user's work. By definition data can be considered information only when it is relevant in a given context. This is, however, not constrained to information. Retrieved relevant data can also function as knowledge of the individual or the organisation when it is used [1].

Content is usually handled by Content Management Systems (CMSs). Most definitions focus on the functionality of these systems, just like the following one, stating that a CMS is a:

> 'system for the creation, modification, archiving and removal of information resources from an organised repository. Includes tools for publishing, format management, revision control, indexing, search and retrieval' [3]

CMS vs. WCMS. The term Content Management System is often confused with Web Content Management System (WCMS). They share a common root, moreover both are content management systems, but WCMSs concentrate on managing the content of web portals, while CMSs are managing more general content. So Web Content Management Systems are Content Management Systems with more specific structure and content description formalisms.

CMS Architecture. CMSs usually consist of three major parts: a content creation, a content management and a presentation subsystem, as it can be observed in Figure 1. The content creation part is responsible for managing inputs, in other

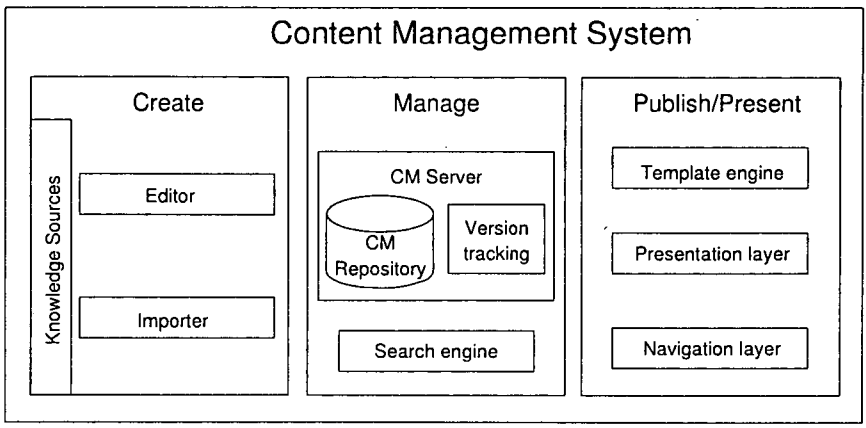


Figure 1: Content Management System Architecture

words gathering content from various knowledge sources. The task of the management part is to store content, together with appropriate versioning information; to ensure access rights management; and to offer information retrieval functions, such as full-text search. The presentation part displays or prints the content using different media types, such as computer screens, mobile devices or printed paper. This part of a CMS has a template engine offering patterns for representing content, adjusting it to the desired use. The presentation layer also handles user navigation on the pages of content – although this function is more relevant in the case of a web content management systems.

Characteristics of CMS. Ideally, the role of a content management system is to be the nerve centre of an enterprise information infrastructure [2, p.7]. It should aggregate data and information from various sources and deliver these to the appropriate recipients. In reality, there are lots of information sources in an organisation, but most of these, such as laptops of individuals or the heads of employees, are not directly connected to the content management system.

In the information system approach, content management is more than data management. For managing pure data, very sophisticated and well-developed tools are available, namely database management systems (DBMSs). However, DBMSs reach their limits quite soon, when used to process content, since DBMSs are designed to manage well-structured data. Content, in contrast, is unstructured with some more-or-less defined meta-data descriptors. This requires an approach completely different from that of data management.

Commoditisation. As a Gartner-study demonstrated in 2002, content management is not a stand-alone product any more [4]. It is integrated with other enterprise information infrastructure components into larger systems, such as Smart Enterprise Suites or corporate portals. Use of CMSs is not a competitive advantage any

more, but a necessity. At the same time it is very interesting to see that, in the case of content management, the reason for commoditisation[2] was not emerging technology standards but its simplicity. Technology standardisation is still under evolution. An example is the standard 'JSR-170 Content Repository for Java Technology API' [13], which describes a standard interface for content management to be used in Java-based systems.

2.2.3 Getting the Right Information

Why is content management so relevant? What is the hidden value in content that made the industry so focused on developing content management systems and then later on developing enterprise applications using content? The answer is simple: it adds further value to corporate assets as a very potent source of knowledge.

Content and knowledge have a strong correlation; they complement and overlap each other in many ways [11]. Content is a storage of relevant data and information, which can be used and reused in a corporate problem context.

The question is how to extract knowledge from the vast amount of stored content. Information overload is a common problem in the field of content management. New content can be created in many different forms, additionally numerous distinct resources can be used for this purpose. These pieces of content can be used and reused many times, thus becoming part of the corporate knowledge body.

On the other hand, existing knowledge can be codified. In other words, knowledge can be expressed in an explicit form and put in the organisational memory as content.

Vast amounts of data, information and knowledge are stored as content in a content management system, so the real problem an organisation has to face is how to use them. As mentioned above, decreasing content creation costs cause an increase in the costs of content processing, especially if the latter requires human work. A solution can be the development of content management systems, which provide facilities for effective information retrieval. The keyword is relevance, especially relevance to that business context in which the user is working. The primary challenge in content management is to deliver appropriate content and information to users. Current content management systems provide full-text searching facilities or categorisation for enabling the delivery of relevant information. However, still too many results are provided by these solutions, meaning that the applied methods are not distinctive enough.

3 Semantic-enhanced Content Management

The main purpose of introducing semantics in content management is to retrieve relevant information. Employing a semantic-enhanced content management system (SCMS) can deliver more appropriate information than the currently used methods of classifying information stored in content. As a prerequisite for making a content

[2] Commoditisation is the process by which goods become mass products.

management system semantic-enabled one has to build a domain ontology in the background[3]. This means that the knowledge of a specific domain is processed into a structure representing both the concepts and the relationships between these concepts. Of course, the larger is the ontology, the more content can be described using it.

3.1 Content Processing in a SCMS

Having formalised the knowledge of the domain in the form of an ontology, any content that is fed into the content management system can be annotated with the concepts of the ontology. Concepts can be assigned to the content in general, or parts of the content – for example, by marking words or phrases in the text.

A problem of annotation is that it requires human intelligence in most of the cases. Currently, there are no applications or methods providing a reliable automatic way of revealing relationships between words or phrases in a text and a structured ontology. However, there are promising research activities aiming at this goal.

Annotation is a formalisation of the content, meaning a formal description of what is depicted in the content itself. As soon as the content is annotated, not only the usual descriptive meta-data (such as the meta-data specified by the Dublin Core standard [7]) are known, but, through the structure of the ontology, the meaning of the content becomes known as well. However, it must be stated, that this description relates only to one or some domains – it cannot be assumed that the whole world can be described in a single ontology in finite time.

Figure 2 summarises this process of connecting ontologies and content management systems, using an example in the legal domain. The same process can be applied to other domains as well.

In the following subsections semantic description of content is compared to classical solutions.

3.1.1 Semantic-Enabled CMSs vs. Document Management Systems

The role of Document Management Systems (DMSs) is to handle textual documents and sometimes other forms of documents. They usually treat these documents as black boxes, using only Dublin Core meta-data (such as author, title, creation date, etc.) for their description. Generally DMSs provide indexing and full-text searching facilities as well. In the semantic-enabled content management approach the domain is represented by an ontology and through this representation the content of a document can also be formalised. This means that the system has a certain knowledge of what the document is about, thanks to all those relationships which have already been established between the document and the concepts of the ontology. This information can also be used for inference or searching.

[3]The author presumes a basic knowledge of ontologies. Please refer to [8], [9], [5], [18] and [17] for more information about this topic.

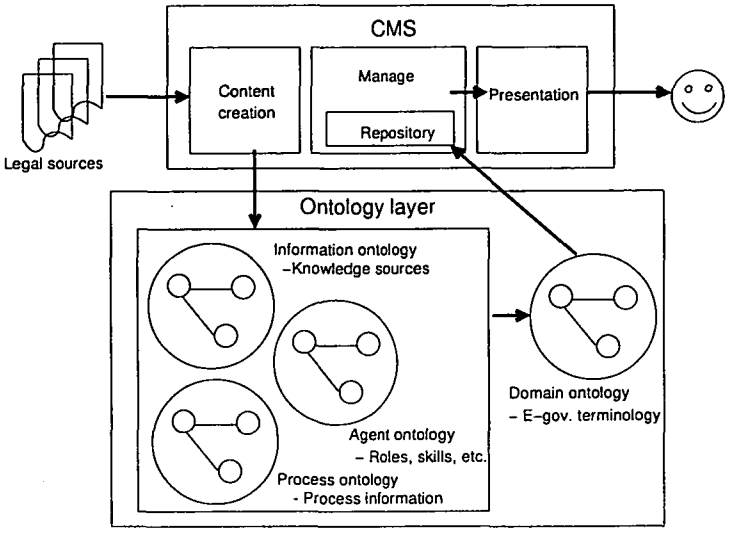


Figure 2: Operation of SCMS in the legal domain

3.1.2 Semantic-Enabled CMSs vs. Full-Text Searching

Full-text searching uses a word index of content as the source of its queries. Ranking of results is determined by different algorithms based on the frequency of words being searched. This is one of the most effective approaches of information retrieval used today. However, it has some drawbacks as well. Certain words and expressions may have one or more synonyms. When searching for a word with synonyms, those contents which only contain the synonyms may not be found. Homonyms are problematic as well, since full-text search cannot distinguish the different meanings of words, making the search results noisy.

Ontology-based content searching means that a concept can be searched independently from its linguistic representation. Actually, word forms or expressions are only instances of a certain concept in the ontology. When a concept is looked up, it will be found regardless of its actual appearance.

This feature of semantic-enabled CMSs can be easily demonstrated by a simple example from the field of Value Added Tax (VAT) regulations. In this context the concept of *natural person* is referred to by the word *customer*. Of course, this word can mean both natural and juristic persons in VAT regulations. When one is looking up content related to the phrase *natural person*, using a semantic-enabled CMS, appropriate parts of the VAT law are expected to be found. This is in spite of the fact that, in the VAT law, the phrase *natural person* is referred to by the expression *customer*.

3.1.3 Semantic-Enabled CMSs vs. Categorisation

Categorisation can be a very good approach to organise and retrieve information. It might have many forms, the simplest of which is the *single categorisation* approach. This is analogous to the folder structure of a hard disk, available on almost every operating system of modern computers, which provides a basic environment for organising files. The problem with this approach is that it uses only a single type of logic in content organisation. It is very hard to find appropriate content if the query follows a different type of logic than that used in the categorisation.

Another approach is multiple categorisation. This uses several terms, and is nothing but the very popular tagging scheme. This approach can ensure quite rich content organisation, however, it can also produce noise in the search results. A drawback is that the person designing the categorisation has to think of all possible aspects of search queries to ensure proper information retrieval. Thus, in most cases, huge amounts of search results are returned, because too many tags are assigned to the content. If sub-categories are used within multiple categorisation, the same problems will arise as in the case of single categorisation.

In the semantic approach the ontology is independent from the content. Thus content is not described by words, but, instead, domain concepts (elements of the ontology) are assigned to parts of the content. When performing information retrieval, the relationships between the concepts in the ontology can be effectively exploited. For example, in some content, such as a document requesting a passport, there is no mention of natural persons. In the corresponding ontology, however, a relationship between a natural person and a passport is defined, namely that only a natural person can request a passport, and a passport can only be assigned to a natural person (both directions can be covered). Even if this relationship is not explicit in any document, the relationship between the natural person and the passport can be discovered and exploited using the domain knowledge formalised in the ontology.

4 Realization in a Pilot Project

In this section we present an application of the principles of semantic-enabled content management systems described above, in the context of a European Union project the author is participating in.

The SAKE project[4] is an ongoing European Union project aiming at the support of knowledge-intensive and formalised processes of a public organisation in a semantic-enabled manner. This system incorporates three major, publicly available

[4]SAKE – Semantic-enabled Agile Knowledge-based eGovernment (IST 027128) is a research project pursued by an international consortium of partners, and co-financed by the 6th EU Framework Programme for Research and Technological Development. The SAKE project commenced on the 1st of March 2006 and lasts for 36 months. The author is leading the development of the Semantic-enabled content management system, one of the most crucial components of the project, which is the responsibility of Corvinus University of Budapest. For more information, please refer to http://www.sake-project.org.

components realizing content, groupware and workflow management functions via integrating existing open-source systems. Additionally, a semantic layer – including an ontology management system and its support tools – has been developed to capture all kinds of semantic information that are provided by components of the system.

4.1 Architecture

From an architectural point of view, the system is built using the classical three-tiered architecture, involving a Presentation Tier, a Business Tier and an Integration Tier, as presented in Figure 3.

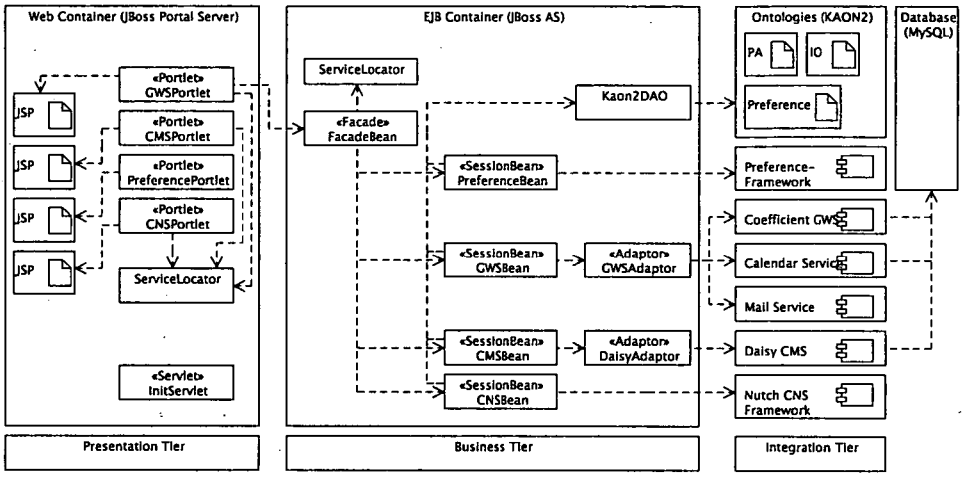


Figure 3: SAKE System Architecture

The *Presentation Tier* is responsible for displaying the content to the user, possibly annotated with additional information. Furthermore, the Presentation Tier handles user interactions using controllers, according to the Model-View-Controller (MVC) paradigm.

The *Business Tier* contains the business logic, realised by the system components mentioned earlier, such as content management or the groupware support modules. A crucial responsibility of the components in this tier is the addition of semantic functionality to the functions provided by the Integration Tier. The Java class FacadeBean provides a unified interface for components in this tier.

The *Integration Tier* consists of various open-source systems on top of which the SAKE System has been built. These include the Daisy content management system; the Coefficient groupware system and its supplementary systems (calendar and mail components); the Preference Framework for handling user-defined preference rules; and the Nutch change notification framework, which is basically a web crawler and a notification system used by the Change Notification System (CNS)

component. Integration of these components is eventually realised in the business logic tier, using adaptor components. Adaptors implement the Data Access Object (DAO) design pattern, decoupling the supporting system functions in the Integration Tier from the business logic in the Business Tier. Adaptors provide therefore a homogeneous way of accessing component functionality by applying a translation between specific SAKE APIs and the APIs of components in the Integration Tier. This approach makes it possible to attach to SAKE an arbitrary software system providing the necessary functionality. Thus the SAKE System can be used in existing public administration or corporate environments, serving as a semantic integration layer built on top of legacy systems.

The *Ontology* component, within the Integration Tier, plays a special role as the storage and reasoning facility for semantic information. It contains various ontologies, such as the Public Administration (PA) ontology for capturing organisational information of Public Administration; the Information ontology (IO) representing meta-data of information sources; and the Preference ontology containing preference rules and data. The Ontology component is attached to the business logic components by a special adaptor for the KAON2 reasoner used in the project.

Business components store semantic data and information in the ontologies while storing system-specific data in the supporting systems in the Integration Tier. Business components are realised as Enterprise JavaBeans (EJBs, or beans in Figure 3), all having an appropriate back-end component in the Integration Tier.

The Presentation Tier consists of Java Server Pages (JSP) descriptor pages, constituting the user interface, and portlets, implementing control logic. Portlets govern the page flow of user interfaces and transmit user and context information to the business components. After processing the user request in the Business Tier, results are delivered to the portlets and presented by JSP pages.

4.2 System Operation

During the operation of the SAKE System, semantic aspects of user interactions are captured as presented in Figure 4.

When users work in a formalised process, they use the Workflow component of the system. The flow of activities is defined in the Process Ontology and is utilised by the workflow engine. Every activity involves some user activities in the system: entering data, creating documents, participating in discussions, etc. The first aspect of semantic information captured by the system is the activity that the user is working in. This state is called the 'Business Context', in the terminology of the system.

As the processes in the system are defined in an ontology, working on a specific process involves the creation of an appropriate instance of a process class in the Process Ontology. This process instance is recording all the user activities: the values she submitted, references the documents she created, and so on.

On the other hand, the user can decide to leave the formalised environment for some reasons, and use other, more knowledge-intensive functions directly in the system. Users can decide to look for documents in the content management system,

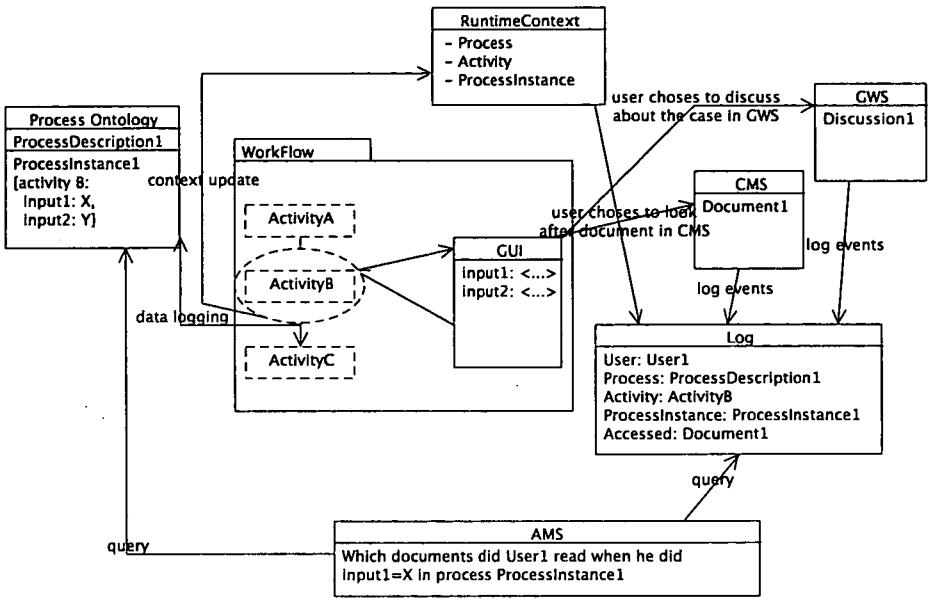


Figure 4: SAKE System Operation

or to discuss a topic in the groupware system. In such cases, the appropriate system component is responsible for recording the user activities.

4.2.1 Capturing Semantics

Various system components record activities by using appropriate ontologies. A basic principle of the system is that everything happening there should be mirrored by the ontologies as well. For example, in the case of Content Management System, when a document is created it is not only stored in the Content Management System in the Integration Tier, but, simultaneously, a new instance of the class Document is created in the Information Ontology. This process, however, provides only a static view as it does not cope with time. The time aspect of user interactions is recorded by the Logging Ontology, by generating new instances of some specific classes of this ontology, such as AccessEvent or CreateEvent, which describe the event. These instances are connected to all other relevant instances in the ontology, such as the Business Context, the User or the Document instances.

These facilities are also used when the user works in the formalised workflow environment, which operates as a kind of marshaller, using various functions of system components. This way all kinds of semantic information are gathered in specific ontologies as the system is running.

4.2.2 Using Semantics

In the previous subsection the way of capturing semantics has been demonstrated. In this process, semantic information about all user activities is stored in various ontologies. However, capturing this information is not enough: it has to be used to satisfy the goals of the system, namely to deliver more precise, more relevant information for its users.

Extracting relevant information of the stored semantic data is the task of the fourth component, the *Attention Management System* (AMS). This component employs a reasoning engine operating on the ontologies. The engine functions by applying user-defined preference rules. These can be either predefined, for example, information about changes in the local regulations, or queried in an ad-hoc way, using complex search expressions. Any data or relationship which can be found in the ontology can be defined in preference rules. This complexity enables the precise description of the user needs.

The reasoning capabilities of the Attention Management System are also used by other components, in the form of related documents or discussions, for example. This component is a crucial element of the system delivering added value to the users. The Attention Management System informs the users about the results of executing predefined preference rules. This is done at user login time, and also in the form of an RSS (Really Simple Syndication) feed, as a notification independent from the SAKE System.

5 Further Research Questions

One of the crucial points of system operation is the proper annotation of the stored documents. Currently, this task is carried out by humans. This is a huge burden on the human resources of the organisation. However, there is no fully reliable method to perform this task using automated systems. In this field, natural language processing and text mining applications show very promising approaches.

The system can be extended to deal with other application fields, involving content types gaining popularity nowadays, such as video or audio clips and pictures. The difficulties encountered in these areas are similar to those appearing in the case of textual documents.

Performance considerations are very exciting and topical problems affecting the usability of the application. Currently, ontology management and reasoning engines are quite slow compared to the well-established database management software, mentioned before. On one hand, this is acceptable since these engines perform complex operations on possibly huge amounts of data. However, the system should remain responsive and fast enough, if it is to be applied by the users in real life situations. Research on improving the speed of reasoning engines shows very promising results, building on techniques learnt from the field of database management, as it is shown, for example, in [12].

6 Conclusions

Current state-of-the-art techniques employed in CMSs are not sufficient enough to handle the the vast amounts of information created and used in an organisation in the course of everyday work. CMSs are not able to effectively manage constantly changing and expanding laws and regulations, which are crucial in public administration.

A major problem is that information in CMSs is stored in an unstructured way. Due to this problem the retrieval of information is also less effective. Moreover general information retrieval algorithms (such as full-text search or categorisation) do not provide results relevant enough.

A solution, as presented in this paper, can be the systematisation and formalisation of domain knowledge. Ontologies provide a formal representation of the given domain, which can be used for mapping content onto the conceptual structure of the domain.

At the same time, building an ontology is far from being a task easy to accomplish. Thus, as a prerequisite for the use of semantic-enabled technologies, large investment of work is needed in the formalisation of a domain. However, the development of adequate ontologies helps solving numerous further tasks. The return on investment is thus realised not only in the field of information retrieval, but also in other areas, such as system and application development, communication, and the operation of the organisation in general.

References

[1] Ackoff, R. L. From data to wisdom. *Journal of Applied Systems Analysis*, 16:3–9, 1989.

[2] APQC. Managing content and knowledge. Report on survey, APQC, 2001. http://www.apqc.org/portal/apqc/ksn?paf_gear_id=contentgearhome &paf_dm=full&pageselect=detail&docid=106067.

[3] Browne, G. and Jermey, J. *Website Indexing (2nd edition)*. Auslib Press, 2007.

[4] Caldwell, F., Gilbert, M., and Hayward, S. CM, portals and collaboration fading: Enter 'smart' suite. Letter from the editor, Gartner, 2002.

[5] Corcho, O., Fernández-López, M., and Gómez-Pérez, A. Methodologies, tools and languages for building ontologies. where is their meeting point? *Data & Knowledge Engineering*, 46:41–64, 2003.

[6] Doyle, B. Seven stages of the CM lifecycle. *EContent*, 2005. http://www.ecmag.net/Articles/ArticleReader.aspx?ArticleID=13554.

[7] Dublin Core Metadata Initiative. ANSI/NISO Z39.85, the Dublin Core metadata element set. Specification, Dublin Core Metadata Initiative, 2007. http://www.niso.org/standards/standard_detail.cfm?std_id=725.

[8] Gomez-Perez, A. Ontological engineering: A state of the art. *Expert Update*, 2(3):38–43, 1999.

[9] Gruber, T. R. A translation approach to portable ontology specifications. *Knowledge Acquisition*, 5(2):199–220, 1993.

[10] Houghton Mifflin Company. *The American Heritage Dictionary of the English Language.* Houghton Mifflin Company, fourth edition, 2004. Answers.com, http://www.answers.com/topic/content.

[11] Kő, A. and Gábor, A. Content industry and knowledge management. In Remenyi, Dan, editor, *Proceedings of the 3rd European Conference on Knowledge Management (ECKM)*, pages 255–261. Academic Conferences Limited, Reading, UK, 2002.

[12] Lukácsy, G. and Szeredi, P. Efficient description logic reasoning in Prolog, the DLog system. *Submitted to: Theory and Practice of Logic Programming*, 2008.
http://sintagma.szit.bme.hu/lukacsy/publikaciok/dlog_tplp_submission.pdf.

[13] Nuescheler, D. JSR 170: Content Repository for Java Technology API. Specification, Sun Microsystems, 2005. http://jcp.org/en/jsr/detail?id=170.

[14] Prideaux, R. What does a Content Management System do? *TechSoup*, 2005. http://www.icthubknowledgebase.org.uk/whatdoesacmsdo.

[15] Skidmore, D. What is the difference between transactional and content data in an internet packet? In Warren, Mark, editor, *Fourth Australian Institute of Computer Ethics Conference*, pages 51–59, Deakin University, Geelong Victoria Australia, 2005. Waurn Ponds Campus, Geelong, Deakin University, School of Information Systems.

[16] The Linux Information Project. Content definition, 2005.
http://www.linfo.org/content.html.

[17] Uschold, M. Building ontologies: Towards a unified methodology. In *Proceedings of Expert Systems '96*, Cambridge, December 16-18th, 1996.

[18] Uschold, M. and King, M. Towards a methodology for building ontologies. In *IJCAI-95 Workshop on Basic Ontological Issues in Knowledge Sharing*, Montreal, Canada, 1995.

[19] Wordnet. Content, 2007.
http://wordnet.princeton.edu/perl/webwn?s=content.

Acta Cybernetica 18 (2008) 595–613.

Computer-Based Intelligent Educational Program for Teaching Chemistry

Róbert Pántya* and László Zsakó†

Abstract

Improving problem-solving skills is a basic requirement in present education. Logic programming languages, often used in the area of artificial intelligence and expert systems, are very much suitable for developing problem-solving skills.

This paper presents a computer-based tool for solving different chemical problems (calculation of quantum numbers, description of electron configurations of atoms, determination of oxidation numbers and electronegativity), which uses the logic programming language Prolog. The authors argue that this intelligent educational material does not only improve chemistry education but it also inspires students to create rule-based systems when they face various problems of everyday life.

Keywords: expert systems, rule-based systems, intelligent educational programs, artificial intelligence

1 Introduction

Computer-assisted problem solving is an important field of informatics education. There are countless occasions in everyday life in which people are confronted with complicated tasks. Students have to decide if such problems can be solved using the tools of informatics, or not. If solving a problem requires the use of a computer, they either have to select an existing tool adequate for the task, or have to build a new one [15].

Computers and programs are frequently used in education. However, such educational programs cannot work without intelligent educational modules. Beyond its basics functions, educational software has to provide advice to students. In the centre of an intelligent tutoring system there is an expert system which encapsulates a brief summary of the knowledge of a given topic [3].

Expert systems are used to solve problems that require natural intelligence. Knowledge is stored in expert systems in a way that makes it possible for the system

*Károly Róbert College, Department of Business Mathematics and Informatics, E-mail: rpantya@karolyrobert.hu

†ELU Department of Teacher Training in Computer Science, E-mail: zsako@ludens.elte.hu

to provide a solution for a problem, making decisions when necessary, and justifying its choices using an appropriate argumentation. Besides transforming knowledge into the most appropriate form, the building of an expert systems also involves further elements, such as strategies for knowledge representation and consistency checking, and the use of heuristic search. Such tasks can be implemented efficiently by using artificial intelligence techniques [12].

There are a lot of existing expert systems (Dendral, MetabolExpert, Mycin, Internist, Guidon, MProlog Shell, Hearsey etc. [12]). Expert systems have attracted considerable attention in the field of chemistry over the past three to four decades. Dendral (Dendritic Algorithm), the first successful expert system in chemistry, was constructed in the 1970's. Its primary aim was to identify unknown organic molecules by analyzing their mass spectra and using the knowledge of chemistry. Originally it was developed for being deployed in a satellite in NASA's Mars program [6].

Further famous chemical expert systems include the Metabol Expert (used in chemical, medical and biological predictions), SELEX (which can be used for the evaluation and quantitative rating of data published on selenium content of foods), and others, such as expert systems for structure elucidation, relying on spectroscopy knowledge bases, data property expert systems, etc. [8].

Another interesting expert system, which can be used in teaching chemistry, is PIRExS (Predicting Inorganic Reactivity Expert System) [2], which predicts the products of inorganic reactions.

Rule-based expert systems seem to be most appropriate for solving chemical problems. In relation to this, it may be of interest to study the origins of the periodic table. In the 19^{th} century, D. I. Mendeleyev arranged the chemical elements in a table according to their atomic mass. At that time the atomic structure was unknown, and thus the most important feature of the atoms was their atomic mass. Mendeleyev realised that every eighth element is similar, and therefore he placed such similar elements in the same column.

There were blank places in his table but he trusted that appropriate new elements, which were not yet discovered, would be found later. He could not explain this similarity of chemical properties, as he did not know the atomic structure. But his conclusions were correct. He discovered a wonderful natural law. The way Mendeleyev reasoned is characteristic of rule-based expert systems. 60 years later, after the discovery of protons and electrons, his system could be explained: the order of the elements is determined by the number of protons, while the chemical properties depend on the characteristics of the electron cloud [5].

In a general chemistry course the studies of electron configurations, quantum numbers and the periodic table are basic topics. There are several papers describing techniques for improving the process of learning chemistry. Iza and Gil [10] provide a mnemonic method for assigning the electron configurations to atoms. According to Mabrouk [11] the periodic table can be used as a mnemonic device for remembering electron configurations.

Many students have difficulties in learning the oxidation number model. Holder et al. [9] have developed an intelligent computer tutoring tool for assigning oxidation

numbers. Solving quantum number problems has been examined by Ardac [1]. Birk [4] has also developed a computer program (OXRULES) to test the effect of various rules on the assignment of oxidation numbers.

In our study we present an intelligent educational material, which can be used in teaching chemistry. The program was built using the Prolog programming language, in Win-Prolog version 4.600 [13]. We assume that the reader is familiar with the Prolog language.

Our educational material uses a rule-based system for systematic presentation of the characteristics of chemical elements, such as the electron configuration, the oxidation numbers, and the electronegativity. Currently, it consists of 50 rules and 150 facts only. However, the development of the program is still in progress. We believe that our innovative method of using a logic programming language not only improves the process of teaching chemistry, but it can also inspire students to construct other rule-based systems, when they are confronted with a new problem. Thus we can reach our primary objective, namely the improvement of student problem-solving skills.

Figure 1 shows the main window of the intelligent educational material. When the *Periodic table* button is pressed in the main window, a new window appears, containing the periodic table with 111 elements. This is shown in Figure 2. The other buttons lead to various tutorials, as explained later.

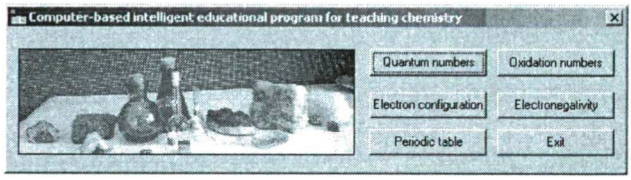


Figure 1: The main window of the intelligent educational material

The rest of the paper is structured as follows. In Section 2 we give a short overview of the basics in chemistry, required for understanding the paper. Next, in Section 3, we discuss the three tutorials developed. The paper is concluded with a brief summary in Section 4.

2 A short overview of chemical basics

An atom is composed of a positively charged nucleus and negatively charged electrons. The nucleus consists of positively charged protons and electrically neutral neutrons. An atom is electrically neutral when the number of protons matches the number of electrons. The number of protons determines the atomic number in the periodic table. The electron cloud consists of electron shells and subshells.

An electron orbital is associated with a region in the space where an electron is likely to be found. The electron configuration of atoms can be described using quantum numbers [7]. There are four kinds of quantum numbers:

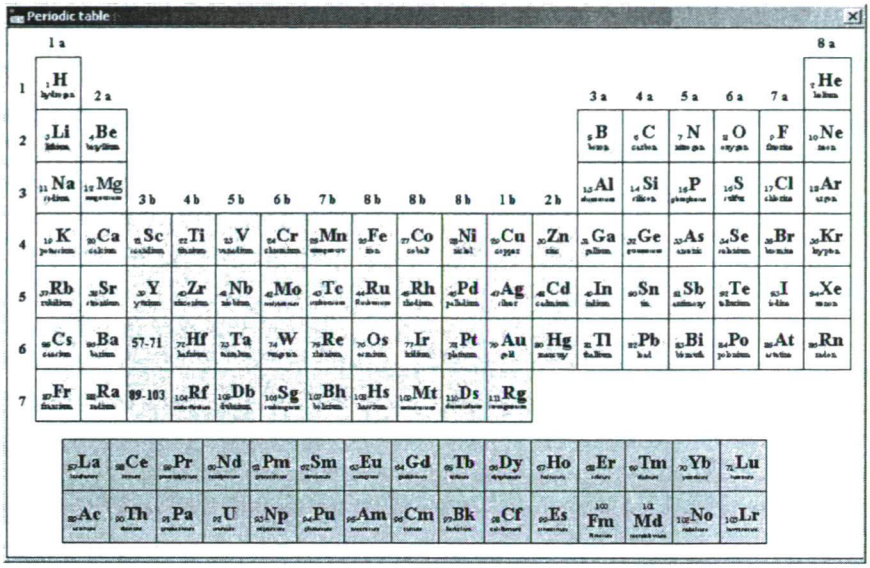


Figure 2: The periodic table, as displayed by the program

- the principal quantum number (n),
- the azimuthal quantum number (l),
- the magnetic quantum number (m_l),
- the spin quantum number (m_s).

The principal quantum number can take the values 1, 2, 3, 4, 5, 6, or 7. The different principal quantum numbers correspond to different shells ($n = 1$ corresponds to the shell K, $n = 2$ to the shell L, etc.; the remaining shells are named M, N, O, P, and Q). The azimuthal quantum number specifies the shape of an atomic orbital. The values of the azimuthal quantum number are $0, 1, \ldots, n-1$, where n is the principal quantum number. Azimuthal quantum numbers are also denoted by letters: $l = 0$ by s, $l = 1$ by p, $l = 2$ by d, $l = 3$ by f, etc. This quantum number also has an orbital meaning, namely it specifies the subshell of the shell [14]. The values of the principal quantum number and the azimuthal quantum number determine the energy level of the electron. Accordingly, the shells and subshells are often referred to as energy levels and sublevels.

The values of the magnetic quantum number can be $-l$, ..., 0, ..., $+l$. The magnetic quantum number determines the energy shift of an atomic orbital due to an external magnetic field. It indicates spatial orientation. The number of the orbitals at a given azimuthal quantum number l is $2 \cdot l + 1$. The maximum number of the orbitals in a shell is n^2, where n is the principal quantum number [5]. Finally,

the spin quantum number is the intrinsic angular momentum of the electron. The values of the spin quantum number are $-\frac{1}{2}$ and $+\frac{1}{2}$.

To determine the electron configuration of an atom we have to use certain rules and principles. According to the Aufbau Principle, electrons fill orbitals starting at the lowest available energy states before filling higher states. Orbitals are generally filled according to the $(n+l)$ rule. This rule states that orbitals with a lower $(n+l)$ value are filled before those with higher $(n+l)$ values. In case of equal $(n+l)$ values, the orbital with a lower n value is filled first.

The Pauli Exclusion Principle states that no two electrons can have the same four quantum numbers. If n, l and m_l are the same m_s must be different, so that the electrons have opposite spins. Therefore a subshell can contain up to $4 \cdot l + 2$ electrons and a shell can contain up to $2 \cdot n^2$ electrons. If multiple orbitals of the same energy are available, Hund's rule says that unoccupied orbitals are filled before occupied orbitals are reused, by electrons having different spins.

The position of an atom in the periodic table is defined by the number of protons of the atom. In the periodic table, rows are called periods and the columns are called groups. There are two types of groups. The primary groups are indicated by the letter '*a*', and the secondary groups are indicated by the letter '*b*'. The periodic table consists of 8 primary groups, 8 secondary groups and 7 periods. The row number is related to the value of the principal quantum number [5].

The valence shell is the outermost shell of an atom. The electrons in the valence shell are referred to as valence electrons. The number of valence electrons determines the number of the primary group and the number of shells determines the number of the period. Valence electrons are important in determining how an element reacts chemically with other elements, i.e. what its chemical behaviour is. In a group (primary or secondary) the number of valence electrons is the same, therefore atoms belonging to the same group have similar chemical properties [5].

The oxidation number is a measure of the degree of oxidation of an atom in a substance. The oxidation number is the real or hypothetical charge that an atom would have if all bonds to atoms of different elements were completely ionic. There are several oxidation number rules. The oxidation number of a free element is zero. In a simple ion the oxidation number is equal to the net charge on the ion. The algebraic sum of oxidation numbers of all atoms in a neutral molecule must be zero.

Electronegativity is a chemical property that describes the power of an atom to attract electrons towards it. Electronegativity cannot be directly measured and must be calculated from other atomic or molecular properties. The most commonly used method of calculation is that originally proposed by Pauling. This gives a dimensionless quantity on a relative scale running from 0.7 to 4.0. In our intelligent program we use these values [5].

3 Tutorials

In this section we discuss three tutorials developed for teaching the principal characteristics of elements.

3.1 Tutorial 1 – Quantum numbers

In this subsection we discuss the features of the program related to answering the following questions:

- What kind of shells are there?
- What kind of subshells has a given shell?
- How many electron orbitals has a given shell?
- How many electron orbitals has a given subshell?
- What is the maximum number of electrons in a given shell?
- What is the maximum number of electrons in a given subshell?
- What kind of magnetic quantum numbers and spin quantum numbers of a given shell and subshell are there, according to the Pauli Exclusion Principle?

To answer the above questions we transform the basics of chemistry, as outlined in Section 2, to the following Prolog facts and rules:

```
principal(1).  principal(2).  principal(3).  principal(4).
principal(5).  principal(6).  principal(7).

azimuthal(0).  azimuthal(1).  azimuthal(2).
azimuthal(3).  azimuthal(4).  azimuthal(5).  azimuthal(6).

magnetic(-6).  magnetic(-5).  magnetic(-4).
magnetic(-3).  magnetic(-2).  magnetic(-1).
magnetic(0).   magnetic(1).   magnetic(2).
magnetic(3).   magnetic(4).   magnetic(5).   magnetic(6).

spin(0.5).     spin(-0.5).

shell(N):-                  principal(N).
subshell(N, L):-            principal(N), azimuthal(L), L<N.
shell_orbitals(N, C) :-     principal(N), C is N*N.
shell_electrons(N, E) :-    principal(N), E is 2*N*N.
subshell_orbitals(L, C) :-  azimuthal(L), C is 2*L+1.
subshell_electrons(L, E) :- azimuthal(L), E is 4*L+2.
quantum(N, L, M, S):-       principal(N), azimuthal(L), L<N,
                            magnetic(M),
                            M>=(-L), M<L+1, spin(S).
```

The first four blocks of Prolog facts list the possible values of the four kinds of quantum numbers (principal, azimuthal, magnetic and spin). Next, the rules for

`shell(N)` and `subshell(N, L)` define the possible shell numbers and the possible pairs of shell and subshell numbers, respectively. Subsequently, given a shell `N` or a subshell `L`, the rules for `shell_orbitals(N, C)`, `shell_electrons(N, E)`, `subshell_orbitals(L, C)`, and `subshell_electrons(L, E)` return the number of corresponding orbitals and electrons. Finally, the rule for `quantum(N, L, M, S)` is capable of listing all allowed combinations of the four kinds of quantum numbers.

The Prolog queries below, when typed in the console window following the `| ?-` prompt, provide the answers to the questions listed at the beginning of the present subsection. The Prolog built-in predicate `fail` is used to enumerate all solutions of a query, while the predicates `write` and `nl` serve for displaying the answer:

```
| ?- shell(J), write(J), nl, fail.
| ?- subshell(3, L), write(3), write(' '), write(L), nl, fail.
| ?- shell_orbitals(3, C).
| ?- shell_orbitals(N, C), write(N), write(' '), write(C), nl, fail.
| ?- shell_electrons(4, E).
| ?- shell_electrons(N, C), write(N), write(' '), write(C),
       nl, fail.
| ?- subshell_orbitals(3, C).
| ?- subshell_orbitals(N, C), write(N), write(' '), write(C),
       nl, fail.
| ?- subshell_electrons(4, E).
| ?- subshell_electrons (N, C), write(N), write(' '), write(C),
       nl, fail.
| ?- quantum(2, 1, M, S), write(M), write(' '), write(S), nl, fail.
| ?- quantum(2, L, M, S), write(L), write(' '),
       write(M), write(' '), write(S), nl, fail.
| ?- quantum(N,L,M,S), write(N), write(' '), write(L), write(' '),
       write(M), write(' '), write(S), nl, fail.
```

There are several of object-oriented extensions of Prolog, such as Visual Prolog, Win-Prolog, etc. These implementations combine the advantages of logic programming and object-oriented programming. As our software for chemistry education was created using Win-Prolog 4.600, we could implement an appropriate graphical interface, which is demonstrated by some examples below.

When the *Quantum numbers* button is pressed in the main window, the panel shown in Figure 3 is displayed.

Next, using the *Shells* button in Figure 3, one obtains the window shown in Figure 4. This window provides important information about the shells. If the *Shells and subshells* button is pressed in Figure 3, then the panel in Figure 5 is presented to the user. Here one can select a shell (e.g. set the principal quantum number to 5). When the *Display* button is pressed in this window, all possible subshell numbers are enumerated in the listbox in the bottom right corner. The values listed are generated by the rule `subshell(N, L)`.

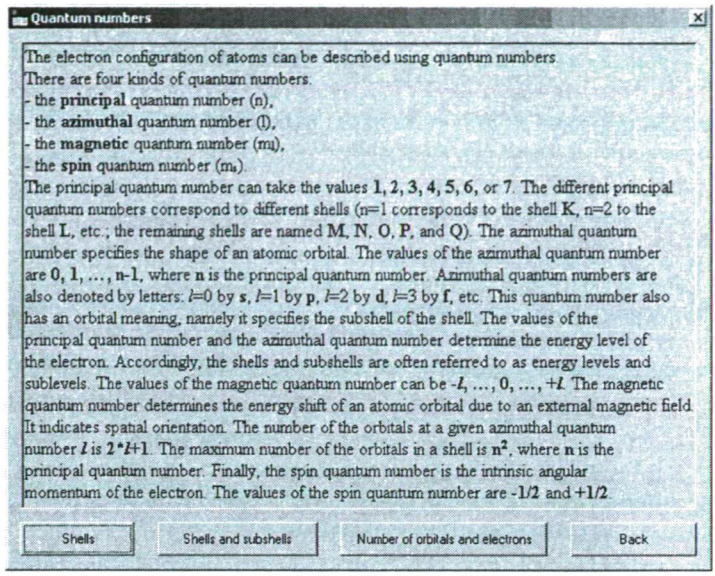


Figure 3: Quantum numbers

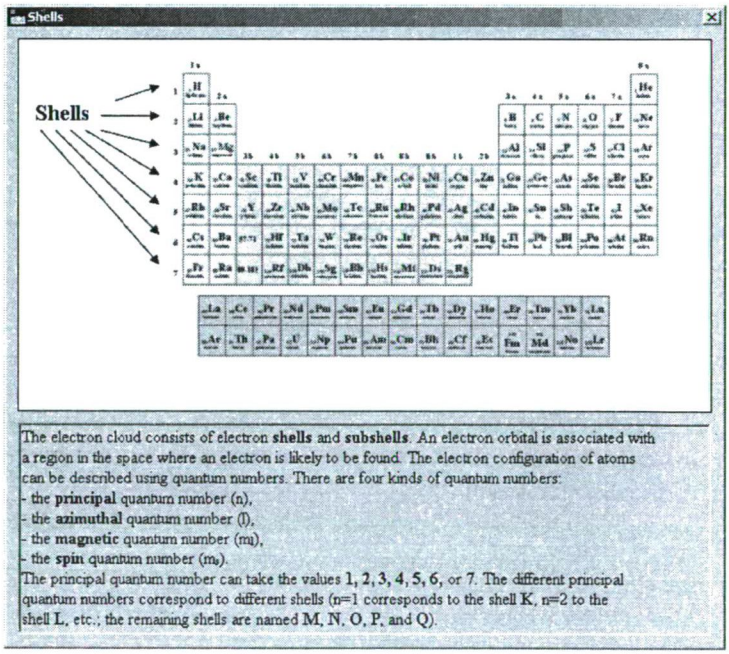


Figure 4: Shells

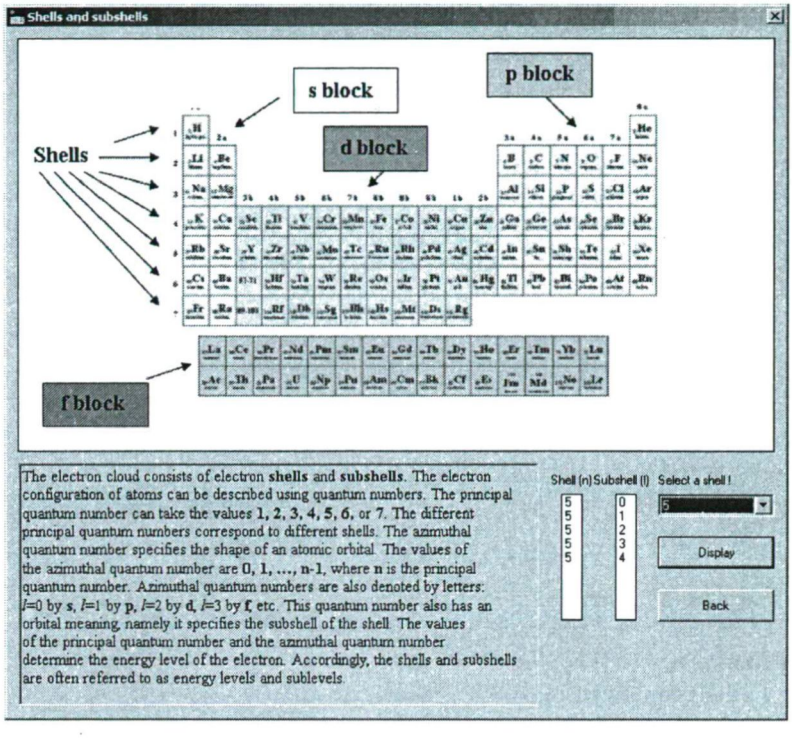


Figure 5: Shells and subshells

Let us return to Figure 3. Using the *Number of orbitals and electrons* button of this window, we get the panel shown in Figure 6. Here, when the user specifies a shell (e.g. by stating that the principal quantum number is 6), and presses the *Display* button, the program enumerates in the listboxes on the left hand side of the panel all possible subshell numbers, and, for each of these, it lists the number of orbitals and electrons. To calculate these possibilities, the program uses the rules `subshell(N, L)`, `subshell_orbitals(L, C)`, and `subshell_electrons(L, E)`.

3.2 Tutorial 2 – Electron configurations of the elements

This subsection deals with obtaining the correct electron configuration of a given element. The program can show which electron orbital is filled, and how many electrons there are in an orbital. The corresponding rules use the Aufbau Principle and the $(n + l)$ rule.

The facts used here are the very same as the ones presented in Section 3.1. The rules are the following:

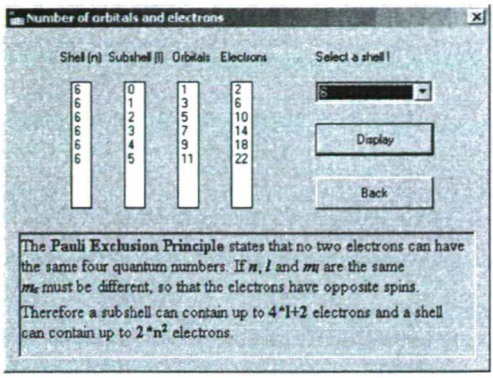


Figure 6: The number of orbitals and electrons

```
electron_orbital(E, N, L):-
        E>0, N<8,
        electron(L, E, E_L, NewE),
        write(N), write(' '), write(L),
        write(' '), write(E_L), nl,
        next_n_l(N, L, NewN, NewL),
        electron_orbital(NewE, NewN, NewL).
electron_orbital(_, N, _):-
        N=8.
electron_orbital(E, _, _):-
        E=0.

electron(L, E, E_L, NewE):-
        E0 is 4*L+2, E>=E0, E_L is E0, NewE is E-E0.
electron(L, E, E_L, NewE):-
        E0 is 4*L+2, E<E0,  E_L is E,  NewE is 0.

next_n_l(N, L, NewN, NewL):-
        L=0, S is N+1, NewL is N // 2, NewN is S-NewL.
next_n_l(N, L, NewN, NewL):-
        L>0, NewN is N+1, NewL is L-1.
```

The predicate `electron_orbital(E, N, L)` has 3 arguments: `E` is the number of electrons, `N` is the principal quantum number, and `L` is the azimuthal quantum number. Its task is to display the correct electron configuration of `E` electrons, starting from the principal quantum number `N` and azimuthal quantum number `L`. To display the complete electron configuration of `E` electrons, the predicate should thus be invoked in the form `electron_orbital(E, 1, 0)`.

The whole predicate can be expressed using three rules. The first rule checks if there are any remaining electrons (`E>0`), and that the principal quantum number

is valid (`N<8`). Next, it calls an auxiliary predicate `electron`, which, given the azimuthal quantum number `L`, splits the number of electrons `E` in two parts: `E_L` is the number of electrons to be placed in the subshell `L`, and `NewE` is the number of remaining electrons (here, `E = E_L+NewE` always holds). Having displayed the triple (`N,L,E_L`), a second auxiliary predicate is invoked: `next_n_l(N, L, NewN, NewL)` receives the pair (`N,L`) and returns the next such pair in (`NewN,NewL`), according to the $(n + l)$ rule. Finally, the predicate is invoked recursively, with the remaining number of electrons, the new shell number, and the new subshell number.

The second and third rule of the predicate `electron_orbital` serve for stopping the recursion, when we run out of shells (`N=8`), or when there are no more electrons to place (`E=0`).

The predicate `electron(L, E, E_L, NewE)` first calculates how many electrons (`E0`) can be in the given subshell `L`. The first rule deals with the case when we have at least `E0` electrons, while in the second rule we have a situation where all the electrons can be placed in the given subshell.

The predicate `next_n_l(N, L, NewN, NewL)` determines the new n and l values using the $(n + l)$ rule. It generates `NewN` from `N`, and `NewL` from `L`. In the first case, when `L=0`, the $n + l$ value increases by 1. In this case `NewL` becomes `N//2` (where `//` is the integer division operator). In the second case, the $n + l$ value does not change: `NewN is N+1` and `NewL is L-1`.

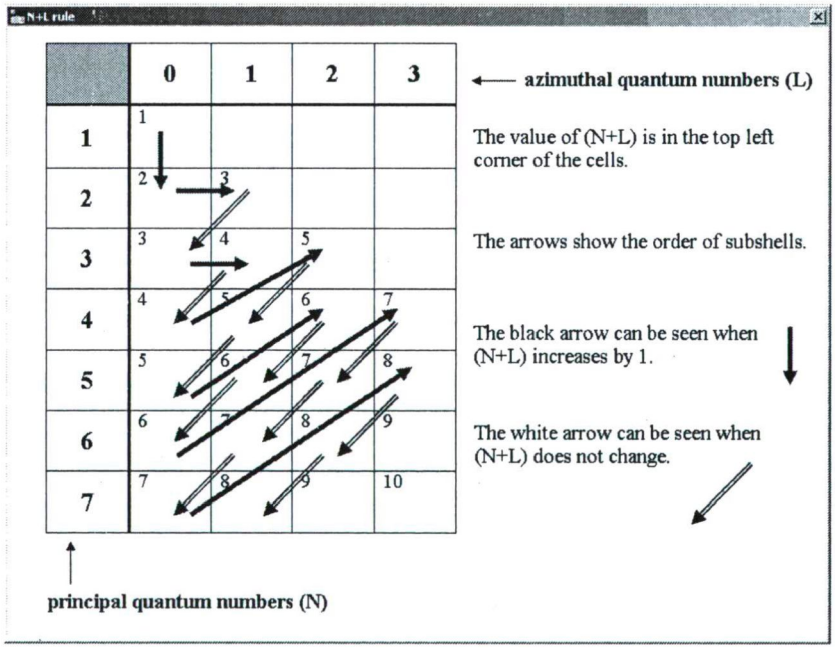


Figure 7: The N+L rule, as displayed by the educational tool

Figure 7 displays the order of subshells, i.e. how the n and l values change in

subsequent invocations of the predicate `next_n_l`.

Figure 8 shows a window for determining the electron configuration. In this particular case the number 80 has been entered, denoting the atomic number, i.e. the number of electrons. When the *Display* button of the window is pressed, the program lists the possible subshells, together with the number of electrons in that subshell, in the bottom left listboxes. It uses the predicate invocation `electron_orbital(E, 1, 0)` to obtain the numbers shown.

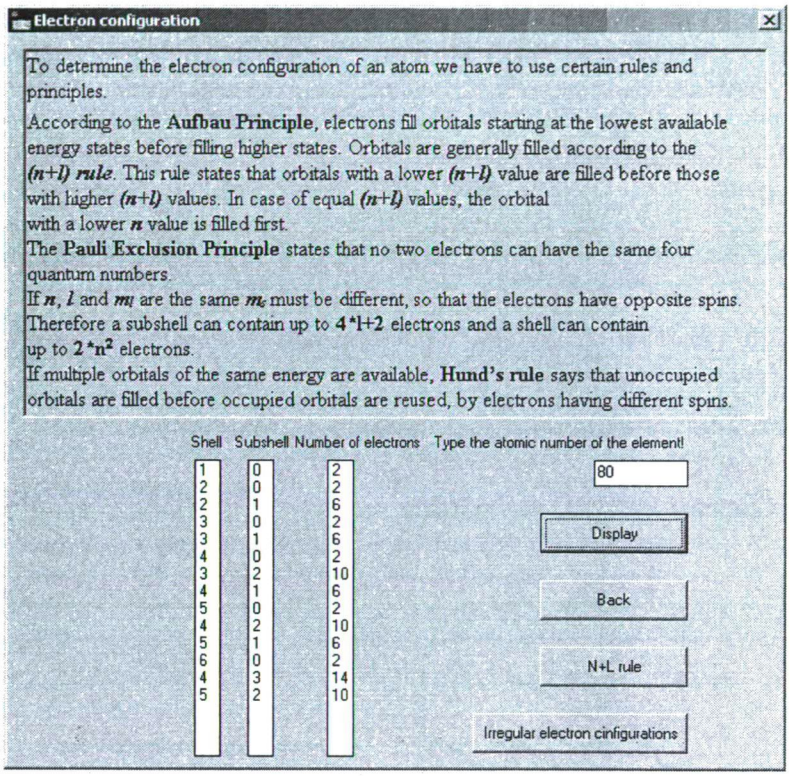


Figure 8: The electron configuration of a given element

There are 20 elements, out of the 111, for which the exact electron configuration cannot be predicted by these rules. The deviation from the regular electron configuration is not significant (only 1 or 2 electrons are in another atomic orbital).

These elements are in the d or f blocks. Some of the irregular electron configurations are predicted on the basis of the stability of half- or wholly filled subshells (for example: Cr, Cu, Mo, Ag or Au). The corresponding configurations are s^1d^5 or s^1d^{10}.

There are some irregular configurations in the f blocks. When atomic orbitals $6s$ and $7s$ are filled, the next electron occupies an orbital d, rather than a $4f$ or a $5f$. But appropriate electrons occupy orbitals f, and orbital d is filled completely

later [16].

To deal with these irregularities, the button labelled *Irregular electron configurations* is provided in the window. When this button is pressed, the program displays a new panel shown in Figure 9. This window shows the elements having an irregular electron configuration.

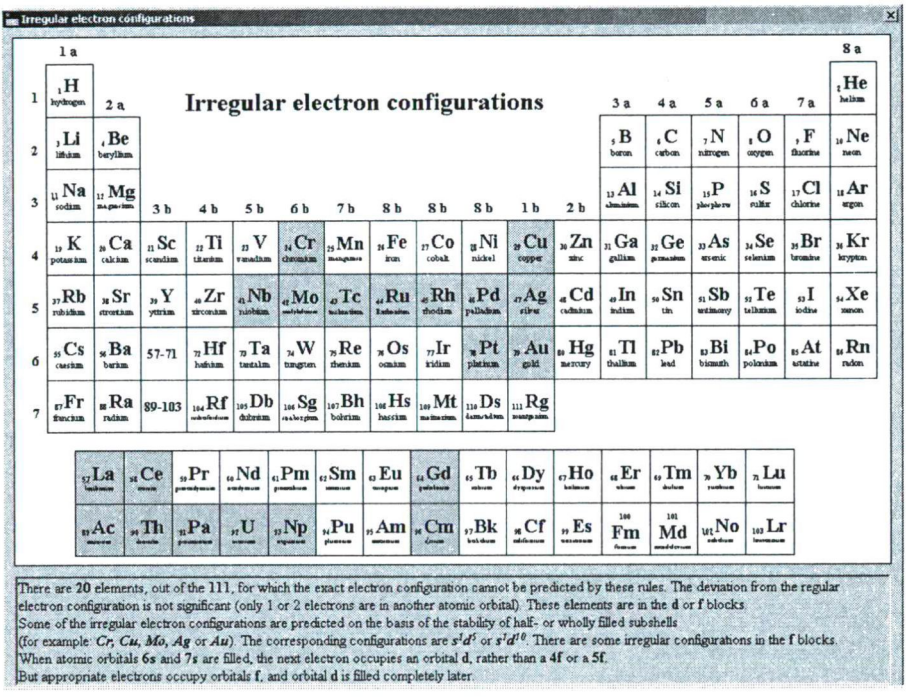


Figure 9: Irregular electron configurations

We believe that it is important that our rule-based system, in addition to determining regular electron configurations, is also able to draw attention to irregular electron configurations. These observations can motivate students to examine such exceptions in more detail.

3.3 Tutorial 3 – Oxidation numbers and electronegativity

In this subsection we show how the program determines the typical oxidation numbers of a given element. Furthermore we discuss questions of the following type:

- Which are the typical oxidation numbers of the elements in a group?
- How much electronegativity has an element?
- If there is a bond between elements, which of the two elements has higher electronegativity?

- How much electronegativity is there in a group?
- How do the electronegativity values change in a period and a group?

To store information on chemical elements, we use the predicate `element`, which has the following 6 arguments:

- the atomic number of the element,
- the name of the element,
- the period number of the element,
- the group number of the element,
- the type of the group: primary (letter '*a*') or secondary (letter '*b*'),
- the value of the electronegativity.

There are 111 known elements in the periodic table. We show only a subset of the 111 facts:

```
element(1, 'hydrogen', 1, 1, 'a', 2.1).
element(2, 'helium', 1, 8, 'a', 0).
element(3, 'lithium', 2, 1, 'a', 1.0).
element(4, 'beryllium', 2, 2, 'a', 1.5).
element(5, 'boron', 2, 3, 'a', 2.0).
element(6, 'carbon', 2, 4, 'a', 2.5).
element(7, 'nitrogen', 2, 5, 'a', 3.0).
element(8, 'oxygen', 2, 6, 'a', 3.5).
element(9, 'fluorine', 2, 7, 'a', 4.0).
element(10, 'neon', 2, 8, 'a', 0).
element(11, 'sodium', 3, 1, 'a', 0.9).
element(12, 'magnesium', 3, 2, 'a', 1.2).
element(13, 'aluminium', 3, 3, 'a', 1.5).
element(14, 'silicon', 3, 4, 'a', 1.8).
element(15, 'phosphorus', 3, 5, 'a', 2.1).
element(16, 'sulfur', 3, 6, 'a', 2.5).
element(17, 'chlorine', 3, 7, 'a', 3.0).
element(18, 'argon', 3, 8, 'a', 0).
element(19, 'potassium', 4, 1, 'a', 0.8).
element(20, 'calcium', 4, 2, 'a', 1.0).
element(21, 'scandium', 4, 3, 'b', 1.3).
element(22, 'titanium', 4, 4, 'b', 1.6).
element(23, 'vanadium', 4, 5, 'b', 1.6).
```

The predicate `oxidation_number`, shown below, has two arguments. The first one is the name of the element and the second one is the possible oxidation number of the element. An element can have several oxidation numbers.

The first rule expresses that for every element, the oxidation number 0 is appropriate. This is because the oxidation number of a free element is zero. Subsequent rules express the relation between the group of the element and the oxidation number.

```
oxidation_number(X, 0):- element(_, X, _, _, _, _).
oxidation_number(X, Y):- element(_, X, _, Y, _, _), Y<6.
oxidation_number(X, Y):- element(_, X, _, Y, 'b', _), Y>5, Y<8.
oxidation_number(X, 3):- element(_, X, _, 5, 'a', _).
oxidation_number(X,-3):- element(_, X, S, 5, 'a', _), S<6.
oxidation_number(X,-2):- element(_, X, S, 6, 'a', _), S<6.
oxidation_number(X, 4):- element(_, X, S, 6, 'a', _), S>2, S<7.
oxidation_number(X,-1):- element(_, X, _, 7, 'a', _).
oxidation_number(X, 1):- element(_, X, S, 7, 'a', _), S>2, S<7.
oxidation_number(X, 5):- element(_, X, S, 7, 'a', _), S>2, S<7.
oxidation_number(X, 3):- element(_, X, 4, O, 'b', _), O>3.
oxidation_number(X, 2):- element(_, X, 4, O, 'b', _), O>4.
oxidation_number(X, 2):- element(_, X, S, 8, 'b', _), S>4, S<7.
oxidation_number(X, 4):- element(_, X, S, 8, 'b', _), S>4, S<7.
```

However, experimental evidence shows that there exist several additional values. These are handled using appropriate Prolog facts, such as the ones listed below.

```
oxidation_number('thallium', 1).   oxidation_number('carbon', -4).
oxidation_number('carbon', 2).     oxidation_number('tin', 2).
oxidation_number('lead', 2).       oxidation_number('nitrogen', 4).
oxidation_number('nitrogen', 2).   oxidation_number('phosphorus', 4).
oxidation_number('oxigen', -1).    oxidation_number('sulfur', 2).
oxidation_number('sulfur', 6).     oxidation_number('selenium', 6).
oxidation_number('tellurium', 6). oxidation_number('polonium', 2).
oxidation_number('chlorine', 3).   oxidation_number('chlorine', 7).
oxidation_number('iodine', 7).     oxidation_number('copper', 2).
oxidation_number('zinc', 1).       oxidation_number('gold', 3).
oxidation_number('mercury', 1).    oxidation_number('vanadium', 4).
```

Building on the predicates `element` and `oxidation_number`, we now define some further predicates, related to the questions listed at the beginning of the present subsection.

The rule for `group_oxnumber(A, B, C)`, shown below, determines the values of oxidation numbers of a given group. The rule for `compare_en(A, B)` can show which element's electronegativity is higher. The rule `electronegativity(X, En)` retrieves the electronegativity of a given element. The two rules `en_period(S, En)` and `en_group(P, Q, En)` produce the values of the electronegativity in a selected group or in a selected period, respectively.

```
group_oxnumber(A, B, C):-
        element(_, X, _, A, B, _),
        oxidation_number(X, C).

compare_en(A, B):-
        element(_, A, _, _, _, X),
        element(_, B, _, _, _, Y), X>Y,
        write(A), nl.
compare_en(A, B):-
        element(_, A, _, _, _, X),
        element(_, B, _, _, _, Y), X=Y,
        write('They have equal electronegativity'), nl.
compare_en(A, B):-
        element(_, A, _, _, _, X),
        element(_, B, _, _, _, Y), X<Y,
        write(B), nl.

electronegativity(X, En):-
        element(_, X, _, _, _, En).

en_group(P, Q, En):-
        element(_, _, _, P, Q, En).

en_period(S, En):-
        element(_, _, S, _, _, En).
```

If one types the following queries into the console window, one can get the answers to some questions of the type listed at the beginning of the present subsection.

```
| ?- group_oxnumber(4, 'a', C), write(C), nl, fail.
| ?- group_oxnumber(1, 'a', C),  write(C), nl, fail.
| ?- compare_en('carbon',  'hydrogen').
| ?- compare_en('hydrogen',  'hydrogen').
| ?- compare_en('bromine',  'chlorine').
| ?- oxidation_number('carbon', C), write(C), nl, fail.
| ?- electronegativity('carbon',  En).
| ?- en_group(5, 'a', En), write(En), nl, fail.
| ?- en_group(3, 'b', En), write(En), nl, fail.
| ?- en_period(5, En), write(En), nl, fail.
```

Figure 10 shows two example questions submitted in the interactive environment: we ask the oxidation numbers of a given element (17 – chlorine) and of a group (3.b).

When you press the *Display* button on the left hand side of the window, the program lists the possible oxidation numbers of a given element in the listbox. For this, it uses the set of `element` facts. When you press the *Display* button on the

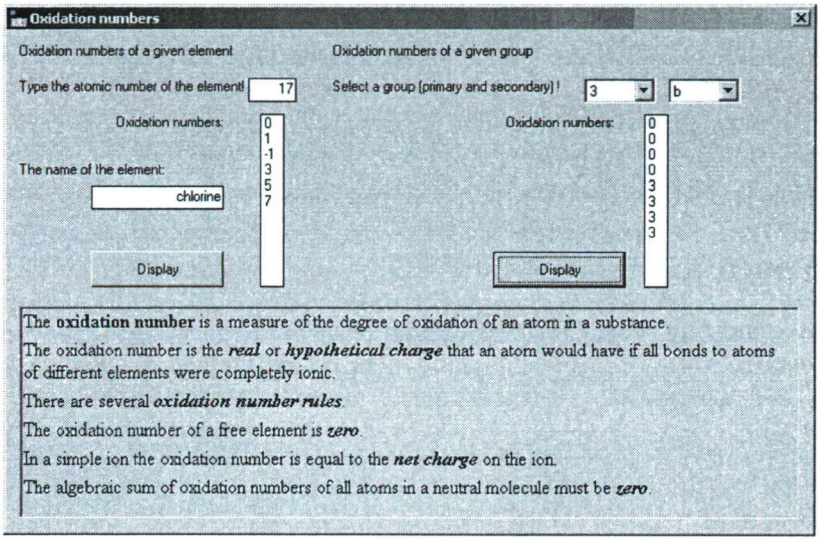


Figure 10: Possible oxidation numbers of a given element and a group

right hand side of the window, the program enumerates in the listbox the possible oxidation numbers of the given group. It uses the `group_oxnumber(A, B, C)` rule for this.

In Figure 11 we present a panel of the program showing the value of the electronegativity of a given element (80 - mercury), of a period (4), and of a group (1.a).

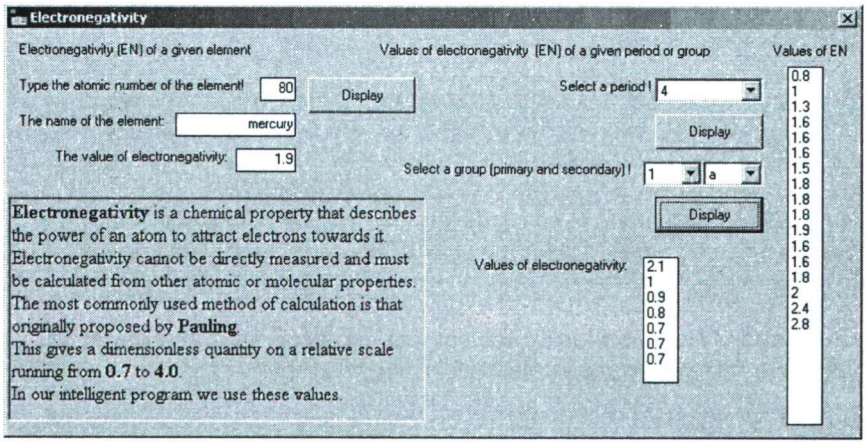


Figure 11: The electronegativity of a given element, period and a group

When you press the *Display* button on the left hand side of the window, the program displays the value of the electronegativity of the given element. It uses the `element` facts. When you press the *Display* button on the right hand side of the window, the program lists the possible values of the electronegativity of a given period and a given group in the listboxes. The `en_period(S, En)` and the `en_group(P, Q, En)` rules are used in this task.

4 Summary

Improving problem-solving skills is a significant aim of education. Logic programming languages are useful tools for computer-assisted problem-solving. Logic programs consist of facts and rules, therefore one can construct rule-based expert systems easily. One can use these tools in chemistry education, too. There are a lot of facts and rules in several subfields of chemistry, learning these is often very difficult for the students. To improve the effectiveness of our educational activities we have to develop new teaching methods. One of such methods is to create (with the active co-operation of the students) a specific rule-based expert system, which contains facts and rules of the given subject.

While constructing the system, the students understand and memorise important facts and rules with much less effort than using traditional learning techniques. If they are confronted with an exception, they are motivated to ask questions, and try to answer these. Thus, we can reach our main objective, namely the improvement of the problem-solving skills of the students.

The development of this educational material is in progress. Future work includes predicting possible chemical bonds and reactions, and determining reaction equations. Our aim is not only to create educational materials, but to build a proper chemical expert system, which also includes an electronic user guide.

References

[1] Ardac, D. Solving quantum number problems: An examination of novice performance in terms of conceptual base requirements. *Journal of Chemical Education*, 79(4):510–513, 2002.

[2] Birk, J. P. *Predicting Inorganic Reactions: The Development of an Expert System in Expert System Applications in Chemistry.* American Chemical Society Symposium Series No.408. ACS Books, Washington, D.C., 1989.

[3] Birk, J. P. The computer as student - an application of artificial intelligence. *Journal of Chemical Education*, 69(4):294–295, 1992.

[4] Birk, J. P. Oxidation number rules: A program to test the effect of various rules on the assignment of oxidation numbers. *Journal of Chemical Education Software*, 6B1, 1993.

[5] Bodonyi, F. *Summary of Chemistry (in Hungarian).* Műszaki Könyvkiadó, Budapest, 1987.

[6] Borgulya, I. *Expert systems, methods and applications (in Hungarian).* ComputerBooks Kiadói Kft., Budapest, 1995.

[7] Brücher, E. *General Chemistry (Structure), educational material (in Hungarian).* KLTE Szervetlen és Analitikai Kémiai Tanszéke, Debrecen, 1992.

[8] Heller, S. R.and Bigwood, D. W. Expert systems in chemistry – applications to data quality. *Proceedings of the 2nd ICIK'87*, pages 99–120, 1987.

[9] Holder, D. A., Johnson, B. G., and Karol, P. J. A consistent set of oxidation number rules for intelligent computer tutoring. *Journal of Chemical Education*, 79(4):465–467, 2002.

[10] Iza, N. and Gil, M. A mnemonic method for assigning the electronic configurations of atoms. *Journal of Chemical Education*, 72(11):1025–1026, 1995.

[11] Mabrouk, S. T. The periodic table as a mnemonic device for writing electronic configurations. *Journal of Chemical Education*, 80(8):894–896, 2003.

[12] Sántáné-Tóth, E. *Knowledge-based Technology, expert systems (in Hungarian).* Dunaújvárosi Főiskola Kiadói Hivatala, Dunaújváros, 2000.

[13] Shalfield, R., Spenser, C., Steel, B., and Westwood, A. *WIN-PROLOG User Guide.* Logic Programming Associates Ltd., London, 2007.

[14] Strong, J. A. The periodic table and electron configurations. *Journal of Chemical Education*, 63(10):834–836, 1986.

[15] Szlávi, P. and Zsakó, L. Informatics as a particular field of education. *Teaching Mathematics and Computer Science*, 3(2):283–294, 2005.

[16] Tőkés, B. and Donáth-Nagy, G. *Chemical lectures and laboratory exercises (in Hungarian).* Scientia Kiadó, Kolozsvár, 2002.

Acta Cybernetica 18 (2008) 615–635.

Factored Value Iteration Converges

István Szita* and András Lőrincz†

Abstract

In this paper we propose a novel algorithm, factored value iteration (FVI), for the approximate solution of factored Markov decision processes (fMDPs). The traditional approximate value iteration algorithm is modified in two ways. For one, the least-squares projection operator is modified so that it does not increase max-norm, and thus preserves convergence. The other modification is that we uniformly sample polynomially many samples from the (exponentially large) state space. This way, the complexity of our algorithm becomes polynomial in the size of the fMDP description length. We prove that the algorithm is convergent. We also derive an upper bound on the difference between our approximate solution and the optimal one, and also on the error introduced by sampling. We analyse various projection operators with respect to their computation complexity and their convergence when combined with approximate value iteration.

Keywords: factored Markov decision process, value iteration, reinforcement learning

1 Introduction

Markov decision processes (MDPs) are extremely useful for formalising and solving sequential decision problems, with a wide repertoire of algorithms to choose from [4, 26]. Unfortunately, MDPs are subject to the 'curse of dimensionality' [3]: for a problem with m state variables, the size of the MDP grows exponentially with m, even though many practical problems have polynomial-size descriptions. Factored MDPs (fMDPs) may rescue us from this explosion, because they offer a more compact representation [17, 5, 6]. In the fMDP framework, one assumes that dependencies can be factored to several easy-to-handle components.

For MDPs with known parameters, there are three basic solution methods (and, naturally, countless variants of them): value iteration, policy iteration and linear programming (see the books of Sutton & Barto [26] or Bertsekas & Tsitsiklis [4] for an excellent overview). Out of these methods, linear programming is generally

*Eötvös Loránd University, Hungary, Department of Information Systems, E-mail: szityu@gmail.com

†Eötvös Loránd University, Hungary, Department of Information Systems, E-mail: andras.lorincz@inf.elte.hu. Please send correspondence to András Lőrincz.

considered less effective than the others. So, it comes as a surprise that all effective fMDPs algorithms, to our best knowledge, use linear programming in one way or another. Furthermore, the classic value iteration algorithm is known to be divergent when function approximation is used [2, 27], which includes the case of fMDPs, too.

In this paper we propose a variant of the approximate value iteration algorithm for solving fMDPs. The algorithm is a direct extension of the traditional value iteration algorithm. Furthermore, it avoids computationally expensive manipulations like linear programming or the construction of decision trees. We prove that the algorithm always converges to a fixed point, and that it requires polynomial time to reach a fixed accuracy. A bound to the distance from the optimal solution is also given.

In Section 2 we review the basic concepts of Markov decision processes, including the classical value iteration algorithm and its combination with linear function approximation. We also give a sufficient condition for the convergence of approximate value iteration, and list several examples of interest. In Section 3 we extend the results of the previous section to fMDPs and review related works in Section 4. Conclusions are drawn in Section 5.

2 Approximate Value Iteration in Markov Decision Processes

2.1 Markov Decision Processes

An MDP is characterised by a sixtuple $(\mathbf{X}, A, R, P, \mathbf{x}_s, \gamma)$, where $\mathbf{X}$ is a finite set of states;[1] A is a finite set of possible actions; $R : \mathbf{X} \times A \to \mathbb{R}$ is the reward function of the agent, so that $R(\mathbf{x}, a)$ is the reward of the agent after choosing action a in state $\mathbf{x}$; $P : \mathbf{X} \times A \times \mathbf{X} \to [0, 1]$ is the transition function so that $P(\mathbf{y} \mid \mathbf{x}, a)$ is the probability that the agent arrives at state $\mathbf{y}$, given that she started from $\mathbf{x}$ upon executing action a; $\mathbf{x}_s \in \mathbf{X}$ is the starting state of the agent; and finally, $\gamma \in [0, 1)$ is the discount rate on future rewards.

A policy of the agent is a mapping $\pi : \mathbf{X} \times A \to [0, 1]$ so that $\pi(\mathbf{x}, a)$ tells the probability that the agent chooses action a in state $\mathbf{x}$. For any $\mathbf{x}_0 \in \mathbf{X}$, the policy of the agent and the parameters of the MDP determine a stochastic process experienced by the agent through the instantiation

$$\mathbf{x}_0, a_0, r_0, \mathbf{x}_1, a_1, r_1, \ldots, \mathbf{x}_t, a_t, r_t, \ldots$$

The goal is to find a policy that maximises the expected value of the discounted total reward. Let the value function of policy π be

$$V^\pi(\mathbf{x}) := E\Big(\sum_{t=0}^{\infty} \gamma^t r_t \;\Big|\; \mathbf{x}=\mathbf{x}_0\Big)$$

[1] Later on, we shall generalise the concept of the state of the system. A state of the system will be a vector of state variables in our fMDP description. For that reason, we already use the boldface vector notation in this preliminary description.

and let the optimal value function be

$$V^*(\mathbf{x}) := \max_{\pi} V^{\pi}(\mathbf{x})$$

for each $\mathbf{x} \in \mathbf{X}$. If V^* is known, it is easy to find an optimal policy π^*, for which $V^{\pi^*} \equiv V^*$. Provided that history does not modify transition probability distribution $P(\mathbf{y}|\mathbf{x}, a)$ at any time instant, value functions satisfy the famous Bellman equations

$$V^{\pi}(\mathbf{x}) = \sum_{a} \sum_{\mathbf{y}} \pi(\mathbf{x}, a) P(\mathbf{y} \mid \mathbf{x}, a) \Big(R(\mathbf{x}, a) + \gamma V^{\pi}(\mathbf{y}) \Big) \tag{1}$$

and

$$V^*(\mathbf{x}) = \max_{a} \sum_{\mathbf{y}} P(\mathbf{y} \mid \mathbf{x}, a) \Big(R(\mathbf{x}, a) + \gamma V^*(\mathbf{y}) \Big). \tag{2}$$

Most algorithms that solve MDPs build upon some version of the Bellman equations. In the following, we shall concentrate on the value iteration algorithm.

2.2 Exact Value Iteration

Consider an MDP $(\mathbf{X}, A, P, R, \mathbf{x}_s, \gamma)$. The value iteration for MDPs uses the Bellman equations (2) as an iterative assignment: It starts with an arbitrary value function $V_0 : \mathbf{X} \to \mathbb{R}$, and in iteration t it performs the update

$$V_{t+1}(\mathbf{x}) := \max_{a} \sum_{\mathbf{y} \in \mathbf{X}} P(\mathbf{y} \mid \mathbf{x}, a) \Big(R(\mathbf{x}, a) + \gamma V_t(\mathbf{y}) \Big) \tag{3}$$

for all $\mathbf{x} \in \mathbf{X}$. For the sake of better readability, we shall introduce vector notation. Let $N := |\mathbf{X}|$, and suppose that states are integers from 1 to N, i.e. $\mathbf{X} = \{1, 2, \ldots, N\}$. Clearly, value functions are equivalent to N-dimensional vectors of reals, which may be indexed with states. The vector corresponding to V will be denoted as $\mathbf{v}$ and the value of state $\mathbf{x}$ by $\mathbf{v}_{\mathbf{x}}$. Similarly, for each a let us define the N-dimensional column vector $\mathbf{r}^a$ with entries $\mathbf{r}^a_{\mathbf{x}} = R(\mathbf{x}, a)$ and $N \times N$ matrix P^a with entries $P^a_{\mathbf{x},\mathbf{y}} = P(\mathbf{y} \mid \mathbf{x}, a)$. With these notations, (3) can be written compactly as

$$\mathbf{v}_{t+1} := \mathbf{max}_{a \in A} \big(\mathbf{r}^a + \gamma P^a \mathbf{v}_t \big). \tag{4}$$

Here, $\mathbf{max}$ denotes the componentwise maximum operator.

It is also convenient to introduce the *Bellman operator* $\mathcal{T} : \mathbb{R}^N \to \mathbb{R}^N$ that maps value functions to value functions as

$$\mathcal{T}\mathbf{v} := \mathbf{max}_{a \in A} \big(\mathbf{r}^a + \gamma P^a \mathbf{v} \big).$$

As it is well known, $\mathcal{T}$ is a max-norm contraction with contraction factor γ: for any $\mathbf{v}, \mathbf{u} \in \mathbb{R}^N$, $\|\mathcal{T}\mathbf{v} - \mathcal{T}\mathbf{u}\|_\infty \leq \gamma \|\mathbf{v} - \mathbf{u}\|_\infty$. Consequently, by Banach's fixed point theorem, exact value iteration (which can be expressed compactly as $\mathbf{v}_{t+1} := \mathcal{T}\mathbf{v}_t$) converges to an unique solution $\mathbf{v}^*$ from any initial vector $\mathbf{v}_0$, and the solution $\mathbf{v}^*$ satisfies the Bellman equations (2). Furthermore, for any required precision $\epsilon > 0$, $\|\mathbf{v}_t - \mathbf{v}^*\|_\infty \leq \epsilon$ if $t \geq \frac{\log \epsilon}{\log \gamma} \|\mathbf{v}_0 - \mathbf{v}^*\|_\infty$. One iteration costs $O(N^2 \cdot |A|)$ computation steps.

2.3 Approximate value iteration

In this section we shall review approximate value iteration (AVI) with linear function approximation (LFA) in ordinary MDPs. The results of this section hold for AVI in general, but if we can perform all operations effectively on compact representations (i.e. execution time is polynomially bounded in the number of variables instead of the number of states), then the method can be directly applied to the domain of factorised Markovian decision problems, underlining the importance of our following considerations.

Suppose that we wish to express the value functions as the linear combination of K *basis functions* $h_k : \mathbf{X} \to \mathbb{R}$ $(k \in \{1, \ldots, K\})$, where $K << N$. Let H be the $N \times K$ matrix with entries $H_{\mathbf{x},k} = h_k(\mathbf{x})$. Let $\mathbf{w}_t \in \mathbb{R}^K$ denote the weight vector of the basis functions at step t. We can substitute $\mathbf{v}_t = H\mathbf{w}_t$ into the right hand side (r.h.s.) of (4), but we cannot do the same on the left hand side (l.h.s.) of the assignment: in general, the r.h.s. is not contained in the image space of H, so there is no such $\mathbf{w}_{t+1}$ that

$$H\mathbf{w}_{t+1} = \mathbf{max}_{a \in A}\left(\mathbf{r}^a + \gamma P^a H\mathbf{w}_t\right).$$

We can put the iteration into work by projecting the right-hand side into $\mathbf{w}$-space: let $\mathcal{G} : \mathbb{R}^N \to \mathbb{R}^K$ be a (possibly non-linear) mapping, and consider the iteration

$$\mathbf{w}_{t+1} := \mathcal{G}\left[\mathbf{max}_{a \in A}\left(\mathbf{r}^a + \gamma P^a H\mathbf{w}_t\right)\right] \tag{5}$$

with an arbitrary starting vector $\mathbf{w}_0$.

Lemma 1. *If $\mathcal{G}$ is such that $H\mathcal{G}$ is a non-expansion, i.e., for any $\mathbf{v}, \mathbf{v}' \in \mathbb{R}^N$,*

$$\|H\mathcal{G}\mathbf{v} - H\mathcal{G}\mathbf{v}'\|_\infty \leq \|\mathbf{v} - \mathbf{v}'\|_\infty,$$

then there exists a $\mathbf{w}^* \in \mathbb{R}^K$ such that

$$\mathbf{w}^* = \mathcal{G}\left[\mathbf{max}_{a \in A}\left(\mathbf{r}^a + \gamma P^a H\mathbf{w}^*\right)\right]$$

and iteration (5) converges to $\mathbf{w}^*$ from any starting point.

Proof. We can write (5) compactly as $\mathbf{w}_{t+1} = \mathcal{G}\mathcal{T}H\mathbf{w}_t$. Let $\widehat{\mathbf{v}}_t = H\mathbf{w}_t$. This satisfies

$$\widehat{\mathbf{v}}_{t+1} = H\mathcal{G}\mathcal{T}\widehat{\mathbf{v}}_t. \tag{6}$$

It is easy to see that the operator $H\mathcal{G}\mathcal{T}$ is a contraction: for any $\mathbf{v}, \mathbf{v}' \in \mathbb{R}^N$,

$$\|H\mathcal{G}\mathcal{T}\mathbf{v} - H\mathcal{G}\mathcal{T}\mathbf{v}'\|_\infty \quad \leq \quad \|\mathcal{T}\mathbf{v} - \mathcal{T}\mathbf{v}'\|_\infty \leq \gamma\|\mathbf{v} - \mathbf{v}'\|_\infty$$

by the assumption of the lemma and the contractivity of $\mathcal{T}$. Therefore, by Banach's fixed point theorem, there exists a vector $\widehat{\mathbf{v}}^* \in \mathbb{R}^N$ such that $\widehat{\mathbf{v}}^* = H\mathcal{G}\mathcal{T}\widehat{\mathbf{v}}^*$ and iteration (6) converges to $\widehat{\mathbf{v}}^*$ from any starting point. It is easy to see that $\mathbf{w}^* = \mathcal{G}\mathcal{T}\widehat{\mathbf{v}}^*$ satisfies the statement of the lemma.

□

Note that if $\mathcal{G}$ is a linear mapping with matrix $G \in \mathbb{R}^{K \times N}$, then the assumption of the lemma is equivalent to $\|HG\|_\infty \leq 1$.

2.4 Examples of Projections, Convergent and Divergent

In this section, we examine certain possibilities for choosing projection $\mathcal{G}$. Let $\mathbf{v} \in \mathbb{R}^N$ be an arbitrary vector, and let $\mathbf{w} = \mathcal{G}\mathbf{v}$ be its $\mathcal{G}$-projection. For linear operators, $\mathcal{G}$ can be represented in matrix form and we shall denote it by G.

Least-squares (L_2-)projection. Least-squares fitting is used almost exclusively for projecting value functions, and the term AVI is usually used in the sense "AVI with least-squares projection". In this case, $\mathbf{w}$ is chosen so that it minimises the least-squares error:

$$\mathbf{w} := \arg\min_{\mathbf{w}} \|H\mathbf{w} - \mathbf{v}\|_2^2.$$

This corresponds to the linear projection $G_2 = H^+$ (i.e., $\mathbf{w} = H^+\mathbf{v}$), where H^+ is the Moore-Penrose pseudoinverse of H. It is well known, however, that this method can diverge. For an example on such divergence, see, e.g. the book of Bertsekas & Tsitsiklis [4]. The reason is simple: matrix HH^+ is a non-expansion in L_2-norm, but Lemma 1 requires that it should be an L_∞-norm projection, which does not hold in the general case. (See also Appendix A.1 for illustration.)

Constrained least-squares projection. One can enforce the non-expansion property by expressing it as a constraint: Let $\mathbf{w}$ be the solution of the constrained minimisation problem

$$\mathbf{w} := \arg\min_{\mathbf{w}} \|H\mathbf{w} - \mathbf{v}\|_2^2, \text{ subject to } \|H\mathbf{w}\|_\infty \leq \|\mathbf{v}\|_\infty,$$

which defines a non-linear mapping $\mathcal{G}_2^c$. This projection is computationally highly demanding: in each step of the iteration, one has to solve a quadratic programming problem.

Max-norm (L_∞-)projection. Similarly to L_2-projection, we can also select $\mathbf{w}$ so that it minimises the max-norm of the residual:

$$\mathbf{w} := \arg\min_{\mathbf{w}} \|H\mathbf{w} - \mathbf{v}\|_\infty.$$

The computation of $\mathbf{w}$ can be transcribed into a linear programming task and that defines the non-linear mapping $\mathcal{G}_\infty$. However, in general, $\|H\mathcal{G}_\infty\mathbf{v}\|_\infty \not\leq \|\mathbf{v}\|_\infty$, and consequently AVI using iteration

$$\mathbf{w}_{t+1} := \arg\min_{\mathbf{w}} \|H\mathbf{w} - \mathcal{T}H\mathbf{w}_t\|_\infty$$

can be divergent. Similarly to L_2 projection, one can also introduce a constrained version $\mathcal{G}_\infty^c$ defined by

$$\mathcal{G}_\infty^c\mathbf{v} := \arg\min_{\mathbf{w}} \|H\mathbf{w} - \mathbf{v}\|_\infty, \text{ subject to } \|H\mathbf{w}\|_\infty \leq \|\mathbf{v}\|_\infty,$$

which can also be turned into a linear program.

It is insightful to contrast this with the approximate linear programming method of Guestrin et al. [14]: they directly minimise the max-norm of the Bellman error, i.e., they solve the problem

$$\mathbf{w}^* := \arg\min_{\mathbf{w}} \|H\mathbf{w} - \mathcal{T}H\mathbf{w}\|_\infty,$$

which can be solved without constraints.

L_1-norm projection. Let $\mathcal{G}_{L_1}$ be defined by

$$\mathcal{G}_1\mathbf{v} := \arg\min_{\mathbf{w}} \|H\mathbf{w} - \mathbf{v}\|_1.$$

The L_1-norm projection also requires the solution of a linear program, but interestingly, the projection operator $\mathcal{G}_1$ is a non-expansion (the proof can be found in Appendix A.1).

AVI-compatible operators considered so far ($\mathcal{G}_2^c$, $\mathcal{G}_\infty^c$ and $\mathcal{G}_1$) were non-linear, and required the solution of a linear program or a quadratic program in each step of value iteration, which is clearly cumbersome. On the other hand, while $G_2\mathbf{v} = H^+\mathbf{v}$ is linear, it is also known to be incompatible with AVI [2, 27]. Now, we shall focus on operators that are both AVI-compatible and linear.

Normalised linear mapping. Let G be an arbitrary $K \times N$ matrix, and define its normalisation $\mathcal{N}(G)$ as a matrix with the same dimensions and entries

$$[\mathcal{N}(G)]_{ij} = \frac{G_{ij}}{\|[HG]_{i,*}\|_\infty}.$$

that is, $N(G)$ is obtained from G by dividing each element with the norm of the corresponding row of HG. All (absolute) row sums of $H \cdot \mathcal{N}(G)$ are equal to 1. Therefore, (i) $\|H \cdot \mathcal{N}(G)\|_\infty = 1$, and (ii) $H \cdot N(G)$ is maximal in the sense that if the absolute value of any element of $\mathcal{N}(G)$ increased, then for the resulting matrix G', $\|H \cdot G'\|_\infty > 1$.

Probabilistic linear mapping. If all elements of H are non-negative and all the row-sums of H are equal, then $\mathcal{N}(H^T)$ assumes a probabilistic interpretation. This interpretation is detailed in Appendix A.2.

Normalised least-squares projection. Among all linear operators, H^+ is the one that guarantees the best least-squares error, therefore we may expect that its normalisation, $\mathcal{N}(H^+)$ plays a similar role among *AVI-compatible* linear projections. Unless noted otherwise, we will use the projection $\mathcal{N}(H^+)$ subsequently.

2.5 Convergence properties

Lemma 2. *Let $\mathbf{v}^*$ be the optimal value function and $\mathbf{w}^*$ be the fixed point of the approximate value iteration (5). Then*

$$\|H\mathbf{w}^* - \mathbf{v}^*\|_\infty \leq \frac{1}{1-\gamma}\|H\mathcal{G}\mathbf{v}^* - \mathbf{v}^*\|_\infty.$$

Proof. For the optimal value function, $\mathbf{v}^* = \mathcal{T}\mathbf{v}^*$ holds. On the other hand, $\mathbf{w}^* = \mathcal{G}\mathcal{T}H\mathbf{w}^*$. Thus,

$$\begin{aligned}
\|H\mathbf{w}^* - \mathbf{v}^*\|_\infty &= \|H\mathcal{G}\mathcal{T}H\mathbf{w}^* - \mathcal{T}\mathbf{v}^*\|_\infty \\
&\leq \|H\mathcal{G}\mathcal{T}H\mathbf{w}^* - H\mathcal{G}\mathcal{T}\mathbf{v}^*\|_\infty + \|H\mathcal{G}\mathcal{T}\mathbf{v}^* - \mathcal{T}\mathbf{v}^*\|_\infty \\
&\leq \|\mathcal{T}H\mathbf{w}^* - \mathcal{T}\mathbf{v}^*\|_\infty + \|H\mathcal{G}\mathbf{v}^* - \mathbf{v}^*\|_\infty \\
&\leq \gamma\|H\mathbf{w}^* - \mathbf{v}^*\|_\infty + \|H\mathcal{G}\mathbf{v}^* - \mathbf{v}^*\|_\infty,
\end{aligned}$$

from which the statement of the lemma follows. For the transformations we have applied the triangle inequality, the non-expansion property of $H\mathcal{G}$ and the contraction property of $\mathcal{T}$. □

According to the lemma, the error bound is proportional to the projection error of $\mathbf{v}^*$. Therefore, if $\mathbf{v}^*$ can be represented in the space of basis functions with small error, then our AVI algorithm gets close to the optimum. Furthermore, the lemma can be used to check *a posteriori* how good our basis functions are. One may improve the set of basis functions iteratively. Similar arguments have been brought up by Guestrin et al. [14], in association with their LP-based solution algorithm.

3 Factored value iteration

MDPs are attractive because solution time is polynomial in the number of states. Consider, however, a sequential decision problem with m variables. In general, we need an exponentially large state space to model it as an MDP. So, the number of states is *exponential* in the size of the description of the task. Factored Markov decision processes may avoid this trap because of their more compact task representation.

3.1 Factored Markov decision processes

We assume that $\mathbf{X}$ is the Cartesian product of m smaller state spaces (corresponding to individual variables):

$$\mathbf{X} = X_1 \times X_2 \times \ldots \times X_m.$$

For the sake of notational convenience we will assume that each X_i has the same size, $|X_1| = |X_2| = \ldots = |X_m| = n$. With this notation, the size of the full state space is $N = |\mathbf{X}| = n^m$. We note that all derivations and proofs carry through to different size variable spaces.

A naive, tabular representation of the transition probabilities would require exponentially large space (that is, exponential in the number of variables m). However, the next-step value of a state variable often depends only on a few other variables, so the full transition probability can be obtained as the product of several simpler factors. For a formal description, we introduce several notations:

For any subset of variable indices $Z \subseteq \{1, 2, \ldots, m\}$, let $\mathbf{X}[Z] := \underset{i \in Z}{\times} X_i$, furthermore, for any $\mathbf{x} \in \mathbf{X}$, let $\mathbf{x}[Z]$ denote the value of the variables with indices in Z. We shall also use the notation $\mathbf{x}[Z]$ without specifying a full vector of values $\mathbf{x}$, in such cases $\mathbf{x}[Z]$ denotes an element in $\mathbf{X}[Z]$. For single-element sets $Z = \{i\}$ we shall also use the shorthand $\mathbf{x}[\{i\}] = \mathbf{x}[i]$.

A function f is a *local-scope* function if it is defined over a subspace $\mathbf{X}[Z]$ of the state space, where Z is a (presumably small) index set. The local-scope function f can be extended trivially to the whole state space by $f(\mathbf{x}) := f(\mathbf{x}[Z])$. If $|Z|$

is small, local-scope functions can be represented efficiently, as they can take only $n^{|Z|}$ different values.

Suppose that for each variable i there exist neighbourhood sets Γ_i such that the value of $\mathbf{x}_{t+1}[i]$ depends only on $\mathbf{x}_t[\Gamma_i]$ and the action a_t taken. Then we can write the transition probabilities in a factored form

$$P(\mathbf{y} \mid \mathbf{x}, a) = \prod_{i=1}^{n} P_i(\mathbf{y}[i] \mid \mathbf{x}[\Gamma_i], a) \tag{7}$$

for each $\mathbf{x}, \mathbf{y} \in \mathbf{X}$, $a \in A$, where each factor is a local-scope function

$$P_i : \mathbf{X}[\Gamma_i] \times A \times X_i \to [0,1] \qquad (\text{for all } i \in \{1,\dots,m\}). \tag{8}$$

We will also suppose that the reward function is the sum of J local-scope functions:

$$R(\mathbf{x}, a) = \sum_{j=1}^{J} R_j(\mathbf{x}[Z_j], a), \tag{9}$$

with arbitrary (but preferably small) index sets Z_j, and local-scope functions

$$R_j : \mathbf{X}[Z_j] \times A \to \mathbb{R} \qquad (\text{for all } j \in \{1,\dots,J\}). \tag{10}$$

To sum up, a factored Markov decision process is characterised by the parameters $\Big(\{X_i : 1 \le i \le m\}; A; \{R_j : 1 \le j \le J\}; \{\Gamma_i : 1 \le i \le n\}; \{P_i : 1 \le i \le n\}; \mathbf{x}_s; \gamma\Big)$, where $\mathbf{x}_s$ denotes the initial state.

Functions P_i and R_i are usually represented either as tables or dynamic Bayesian networks. If the maximum size of the appearing local scopes is bounded by some constant, then the description length of an fMDP is polynomial in the number of variables n.

3.1.1 Value functions

The optimal value function is an $N = n^m$-dimensional vector. To represent it efficiently, we should rewrite it as the sum of local-scope functions with small domains. Unfortunately, in the general case, no such factored form exists [14].

However, we can still approximate V^* with such an expression: let K be the desired number of basis functions and for each $k \in \{1,\dots,K\}$, let C_k be the domain set of the local-scope basis function $h_k : \mathbf{X}[C_k] \to \mathbb{R}$. We are looking for a value function of the form

$$\tilde{V}(\mathbf{x}) = \sum_{k=1}^{K} w_k \cdot h_k(\mathbf{x}[C_k]). \tag{11}$$

The quality of the approximation depends on two factors: the choice of the basis functions and the approximation algorithm. Basis functions are usually selected by the experiment designer, and there are no general guidelines how to automate this process. For given basis functions, we can apply a number of algorithms to determine the weights w_k. We give a short overview of these methods in Section 4. Here, we concentrate on value iteration.

3.2 Exploiting factored structure in value iteration

For fMDPs, we can substitute the factored form of the transition probabilities (7), rewards (9) and the factored approximation of the value function (11) into the AVI formula (5), which yields

$$\sum_{k=1}^{K} h_k(\mathbf{x}[C_k]) \cdot w_{k,t+1} \approx \max_a \sum_{\mathbf{y}\in\mathbf{X}} \Big(\prod_{i=1}^{m} P_i(\mathbf{y}[i] \mid \mathbf{x}[\Gamma_i], a)\Big) \cdot \cdot\Big(\sum_{j=1}^{J} R_j(\mathbf{x}[Z_j], a) + \gamma \sum_{k'=1}^{K} h_{k'}(\mathbf{y}[C_{k'}]) \cdot w_{k',t}\Big).$$

By rearranging operations and exploiting that all occurring functions have a local scope, we get

$$\sum_{k=1}^{K} h_k(\mathbf{x}[C_k]) \cdot w_{k,t+1} = \mathcal{G}_k \max_a \Bigg[\sum_{j=1}^{J} R_j(\mathbf{x}[Z_j], a) + \gamma \sum_{k'=1}^{K} \sum_{\mathbf{y}[C_{k'}]\in\mathbf{X}[C_{k'}]} \Big(\prod_{i\in C_{k'}} P_i(\mathbf{y}[i] \mid \mathbf{x}[\Gamma_i], a)\Big) h_{k'}(\mathbf{y}[C_{k'}]) \cdot w_{k',t}\Bigg] \quad (12)$$

for all $\mathbf{x} \in \mathbf{X}$. We can write this update rule more compactly in vector notation. Let

$$\mathbf{w}_t := (w_{1,t}, w_{2,t}, \ldots, w_{K,t}) \in \mathbb{R}^K,$$

and let H be an $|\mathbf{X}| \times K$ matrix containing the values of the basis functions. We index the rows of matrix H by the elements of $\mathbf{X}$:

$$H_{\mathbf{x},k} := h_k(\mathbf{x}[C_k]).$$

Further, for each $a \in A$, let B^a be the $|\mathbf{X}| \times K$ *value backprojection* matrix defined as

$$B^a_{\mathbf{x},k} := \sum_{\mathbf{y}[C_k]\in\mathbf{X}[C_k]} \Big(\prod_{i\in C_k} P_i(\mathbf{y}[i] \mid \mathbf{x}[\Gamma_i], a)\Big) h_k(\mathbf{y}[C_k])$$

and for each a, define the reward vector $\mathbf{r}^a \in \mathbb{R}^{|\mathbf{X}|}$ by

$$\mathbf{r}^a_{\mathbf{x}} := \sum_{j=1}^{n_r} R_j(\mathbf{x}[Z_j], a).$$

Using these notations, (12) can be rewritten as

$$\mathbf{w}_{t+1} := \mathcal{G}\mathbf{max}_{a\in A}\Big(\mathbf{r}^a + \gamma B^a \mathbf{w}_t\Big). \quad (13)$$

Now, all entries of B^a, H and $\mathbf{r}^a$ are composed of local-scope functions, so any of their individual elements can be computed efficiently. This means that the

time required for the computation is exponential in the sizes of function scopes, but only polynomial in the number of variables, making the approach attractive. Unfortunately, the matrices are still exponentially large, as there are exponentially many equations in (12). One can overcome this problem by sampling as we show below.

3.3 Sampling

To circumvent the problem of having exponentially many equations, we select a random subset $\widehat{\mathbf{X}} \subseteq \mathbf{X}$ of the original state space so that $|\widehat{\mathbf{X}}| = \text{poly}(m)$, consequently, solution time will scale polynomially with m. On the other hand, we will select a sufficiently large subset so that the remaining system of equations is still over-determined. The necessary size of the selected subset is to be determined later: it should be as small as possible, but the solution of the reduced equation system should remain close to the original solution with high probability. For the sake of simplicity, we assume that the projection operator $\mathcal{G}$ is linear with matrix G. Let the sub-matrices of G, H, B^a and $\mathbf{r}^a$ corresponding to $\widehat{\mathbf{X}}$ be denoted by $\widehat{G}$, $\widehat{H}$, $\widehat{B}^a$ and $\widehat{\mathbf{r}}^a$, respectively. Then the following value update

$$\mathbf{w}_{t+1} := \widehat{G} \cdot \max_{a \in A} \left(\widehat{\mathbf{r}}^a + \gamma \widehat{B}^a \mathbf{w}_t \right) \tag{14}$$

can be performed effectively, because these matrices have polynomial size. Now we show that the solution from sampled data is close to the true solution with high probability.

Theorem 1. *Let $\mathbf{w}^*$ be the unique solution of $\mathbf{w}^* = GTH\mathbf{w}^*$ of an FMDP, and let $\mathbf{w}'$ be the solution of the corresponding equation with sampled matrices, $\mathbf{w}' = \widehat{G}T\widehat{H}\mathbf{w}'$. Suppose that the projection matrix G has a factored structure, too. Then iteration (14) converges to $\mathbf{w}'$, furthermore, for a suitable constant Ξ (depending polynomially on n^z, where z is the maximum cluster size), and for any $\epsilon, \delta > 0$, $\|\mathbf{w}^* - \mathbf{w}'\|_\infty \leq \epsilon$ holds with probability at least $1 - \delta$, if the sample size satisfies $N_1 \geq \Xi \frac{m^2}{\epsilon^2} \log \frac{m}{\delta}$.*

The proof of Theorem 1 can be found in Appendix A.3. The derivation is closely related to the work of Drineas and colleagues [11, 12] with the important exception we use the infinity-norm instead of the L_2-norm. The resulting *factored value iteration* algorithm is summarised in Algorithm 1.

4 Related work

The exact solution of factored MDPs is infeasible. The idea of representing a large MDP using a factored model was first proposed by Koller & Parr [17] but similar ideas appear already in the works of Boutilier, Dearden, & Goldszmidt [5, 6]. More recently, the framework (and some of the algorithms) was extended to fMDPs with

Algorithm 1 Factored value iteration with a linear projection matrix G.

% inputs:
% factored MDP, $\mathcal{M} = \left(\{X_i\}_{i=1}^m; A; \{R_j\}_{j=1}^J; \{\Gamma_i\}_{i=1}^m; \{P_i\}_{i=1}^m; \mathbf{x}_s; \gamma\right)$
% basis functions, $\{h_k\}_{k=1}^K$
% required accuracy, ϵ
$N_1 :=$ number of samples
$\widehat{\mathbf{X}} :=$ uniform random N_1-element subset of $\mathbf{X}$
create $\widehat{H}$ and $\widehat{G}$
create $\widehat{B}^a = \widehat{P^a H}$ and $\widehat{\mathbf{r}}^a$ for each $a \in A$
$\mathbf{w}_0 = \mathbf{0}$, $t := 0$
repeat
  $\mathbf{w}_{t+1} := \widehat{G} \cdot \max_{a \in A} \left(\widehat{\mathbf{r}}^a + \gamma \widehat{B}^a \mathbf{w}_t\right)$
  $\Delta_t := \|\mathbf{w}_{t+1} - \mathbf{w}_t\|_\infty$
  $t := t + 1$
until $\Delta_t \geq \epsilon$
return $\mathbf{w}_t$

hybrid continuous-discrete variables [18] and factored partially observable MDPs [23]. Furthermore, the framework has also been applied to structured MDPs with alternative representations, e.g., relational MDPs [13] and first-order MDPs [24].

4.1 Algorithms for solving factored MDPs

There are two major branches of algorithms for solving fMDPs: the first one approximates the value functions as decision trees, the other one makes use of linear programming.

Decision trees (or equivalently, decision lists) provide a way to represent the agent's policy compactly. Koller & Parr [17] and Boutilier et al. [5, 6] present algorithms to evaluate and improve such policies, according to the policy iteration scheme. Unfortunately, the size of the policies may grow exponentially even with a decision tree representation [6, 20].

The exact Bellman equations (2) can be transformed to an equivalent linear program with N variables $\{V(\mathbf{x}) : \mathbf{x} \in \mathbf{X}\}$ and $N \cdot |A|$ constraints:

$$\begin{aligned} \text{maximise:} \quad & \sum_{\mathbf{x} \in \mathbf{X}} \alpha(\mathbf{x}) V(\mathbf{x}) \\ \text{subject to} \quad & V(\mathbf{x}) \leq R(\mathbf{x}, a) + \gamma \sum_{\mathbf{x}' \in \mathbf{X}} P(\mathbf{x}' \mid \mathbf{x}, a) V(\mathbf{x}'), \quad (\forall \mathbf{x} \in \mathbf{X}, a \in A). \end{aligned}$$

Here, weights $\alpha(\mathbf{x})$ are free parameters and can be chosen freely in the following sense: the optimum solution is V^* independent of their choice, provided that each of them is greater than 0. In the approximate linear programming approach, we approximate the value function as a linear combination of basis functions (11),

resulting in an approximate LP with K variables $\{w_k : 1 \leq k \leq K\}$ and $N \cdot |A|$ constraints:

$$\text{maximise:} \quad \sum_{k=1}^{K} \sum_{\mathbf{x} \in \mathbf{X}} w_k \cdot \alpha(\mathbf{x}) h_k(\mathbf{x}[C_k]) \tag{15}$$

$$\text{subject to} \quad \sum_{k=1}^{K} w_k \cdot h_k(\mathbf{x}[C_k]) \leq$$

$$\leq R(\mathbf{x}, a) + \gamma \sum_{k'=1}^{K} w_{k'} \sum_{\mathbf{x}' \in \mathbf{X}} P(\mathbf{x}' \mid \mathbf{x}, a) \cdot h_{k'}(\mathbf{x}'[C_{k'}]).$$

Both the objective function and the constraints can be written in compact forms, exploiting the local-scope property of the appearing functions.

Markov decision processes were first formulated as LP tasks by Schweitzer and Seidmann [25]. The approximate LP form is due to de Farias and van Roy [7]. Guestrin et al. [14] show that the maximum of local-scope functions can be computed by rephrasing the task as a non-serial dynamic programming task and eliminating variables one by one. Therefore, (15) can be transformed to an equivalent, more compact linear program. The gain may be exponential, but this is not necessarily so in all cases: according to Guestrin et al. [14], "as shown by Dechter [9], [the cost of the transformation] is exponential in the induced width of the cost network, the undirected graph defined over the variables $X_1; \ldots; X_n$, with an edge between X_l and X_m if they appear together in one of the original functions f_j. The complexity of this algorithm is, of course, dependent on the variable elimination order and the problem structure. Computing the optimal elimination order is an NP-hard problem [1] and elimination orders yielding low induced tree width do not exist for some problems." Furthermore, for the approximate LP task (15), the solution is no longer independent of α and the optimal choice of the α values is not known.

The approximate LP-based solution algorithm is also due to Guestrin et al. [14]. Dolgov and Durfee [10] apply a primal-dual approximation technique to the linear program, and report improved results on several problems.

The approximate policy iteration algorithm [17, 14] also uses an approximate LP reformulation, but it is based on the policy-evaluation Bellman equation (1). Policy-evaluation equations are, however, linear and do not contain the maximum operator, so there is no need for the second, costly transformation step. On the other hand, the algorithm needs an explicit decision tree representation of the policy. Liberatore [20] has shown that the size of the decision tree representation can grow exponentially.

4.1.1 Applications

Applications of fMDP algorithms are mostly restricted to artificial test problems like the problem set of Boutilier et al. [6], various versions of the SysAdmin task [14, 10, 21] or the New York driving task [23].

Guestrin, Koller, Gearhart and Kanodia [13] show that their LP-based solution algorithm is also capable of solving more practical tasks: they consider the real-time strategy game *FreeCraft*. Several scenarios are modelled as fMDPs, and solved successfully. Furthermore, they find that the solution generalises to larger tasks with similar structure.

4.1.2 Unknown environment

The algorithms discussed so far (including our FVI algorithm) assume that all parameters of the fMDP are known, and the basis functions are given. In the case when only the factorisation structure of the fMDP is known but the actual transition probabilities and rewards are not, one can apply the factored versions of E^3 [16] or R-max [15].

Few attempts exist that try to obtain basis functions or the structure of the fMDP automatically. Patrascu et al. [21] select basis functions greedily so that the approximated Bellman error of the solution is minimised. Poupart et al. [22] apply greedy selection, too, but their selection criteria are different: a decision tree is constructed to partition the state space into several regions, and basis functions are added for each region. The approximate value function is piecewise linear in each region. The metric they use for splitting is related to the quality of the LP solution.

4.2 Sampling

Sampling techniques are widely used when the state space is immensely large. Lagoudakis and Parr [19] use sampling without a theoretical analysis of performance, but the validity of the approach is verified empirically. De Farias and van Roy [8] give a thorough overview on constraint sampling techniques used for the linear programming formulation. These techniques are, however, specific to linear programming and cannot be applied in our case.

The work most similar to ours is that of Drineas et al. [12, 11]. They investigate the least-squares solution of an overdetermined linear system, and they prove that it is sufficient to keep polynomially many samples to reach low error with high probability. They introduce a non-uniform sampling distribution, so that the variance of the approximation error is minimised. However, the calculation of the probabilities requires a complete sweep through all equations.

5 Conclusions

In this paper we have proposed a new algorithm, factored value iteration, for the approximate solution of factored Markov decision processes. The classical approximate value iteration algorithm is modified in two ways. Firstly, the least-squares projection operator is substituted with an operator that does not increase max-norm, and thus preserves convergence. Secondly, polynomially many samples are

sampled uniformly from the (exponentially large) state space. This way, the complexity of our algorithm becomes polynomial in the size of the fMDP description length. We prove that the algorithm is convergent and give a bound on the difference between our solution and the optimal one. We also analysed various projection operators with respect to their computation complexity and their convergence when combined with approximate value iteration. To our knowledge, this is the first algorithm that (1) provably converges in polynomial time and (2) avoids linear programming.

Acknowledgements

The authors are grateful to Zoltán Szabó for calling our attention to the articles of Drineas et al. [11, 12]. This research has been supported by the EC FET 'New and Emergent World models Through Individual, Evolutionary, and Social Learning' Grant (Reference Number 3752). Opinions and errors in this manuscript are the author's responsibility, they do not necessarily reflect those of the EC or other project members.

A Proofs

A.1 Projections in various norms

We wish to know whether $\mathbf{w}_0 = \arg\min_{\mathbf{w}} \|H\mathbf{w} - \mathbf{v}\|_p$ implies $\|H\mathbf{w}_0\|_\infty \leq \|\mathbf{v}\|_\infty$ for various values of p. Specifically, we are interested in the cases when $p \in \{1, 2, \infty\}$. Fig. 1 indicates that the implication does not hold for $p = 2$ or $p = \infty$, only for the case $p = 1$. Below we give a rigorous proof for these claims.

Consider the example $\mathbf{v} = [1, 1]^T \in \mathbb{R}^2$, $H = [1, 2]^T$, $\mathbf{w} \in \mathbb{R}^1$. For these values easy calculation shows that $\|H[\mathbf{w}_0]_{L_2}\|_\infty = 6/5$ and $\|H[\mathbf{w}_0]_{L_\infty}\|_\infty = 4/3$, i.e., $\|H\mathbf{w}_0\|_\infty \not\leq \|\mathbf{v}\|_\infty$ for both cases. For $p = 1$, we shall prove the following lemma:

Lemma 3. *If* $\mathbf{w}_0 = \arg\min_{\mathbf{w}} \|H\mathbf{w} - \mathbf{v}\|_1$, *then* $\|H\mathbf{w}_0\|_\infty \leq \|\mathbf{v}\|_\infty$.

Proof. Let $\mathbf{z} := H\mathbf{w}_0 \in \mathbb{R}^N$. If there are multiple solutions to the minimisation task, then consider the (unique) $\mathbf{z}$ vector with minimum L_2-norm. Let $r := \|\mathbf{z} - \mathbf{v}\|_1$ and let $S(\mathbf{v}, r)$ be the L_1-sphere with centre $\mathbf{v}$ and radius r (this is an N-dimensional *cross polytope* or *orthoplex*, a generalisation of the octahedron).

Suppose indirectly that $\|\mathbf{z}\|_\infty > \|\mathbf{v}\|_\infty$. Without loss of generality we may assume that z_1 is the coordinate of $\mathbf{z}$ with the largest absolute value, and that it is positive. Therefore, $z_1 > \|\mathbf{v}\|_\infty$. Let $\mathbf{e}_i$ denote the i^{th} coordinate vector $(1 \leq i \leq N)$, and let $\mathbf{z}_\delta = \mathbf{z} - \delta\mathbf{e}_1$. For small enough δ, $S(\mathbf{z}_\delta, \delta)$ is a cross polytope such that (a) $S(\mathbf{z}_\delta, \delta) \subset S(\mathbf{v}, r)$, (b) $\forall \mathbf{z}' \in S(\mathbf{z}_\delta, \delta)$, $\|\mathbf{z}'\|_\infty > \|\mathbf{v}\|_\infty$, (c) $\forall \epsilon > 0$ sufficiently small, $(1 - \epsilon)\mathbf{z} \in S(\mathbf{z}_\delta, \delta)$. The first two statements are trivial. For the third statement, note that $\mathbf{z}$ is a vertex of the cross polytope $S(\mathbf{z}_\delta, \delta)$. Consider the cone whose vertex is $\mathbf{z}$ and its edges are the same as the edges of $S(\mathbf{z}_\delta, \delta)$ joining $\mathbf{z}$. It is easy to see that the vector pointing from $\mathbf{z}$ to the origo is contained in this

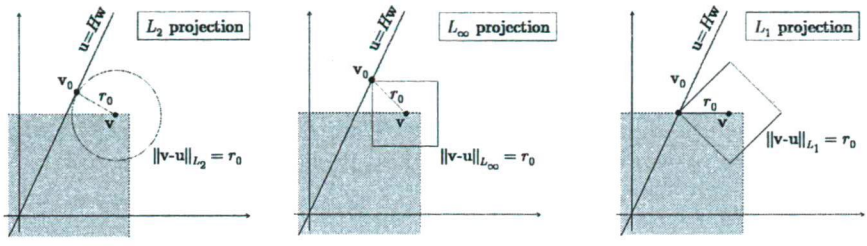


Figure 1: Projections in various norms. The vector $\mathbf{v}$ is projected onto the image space of H, i.e., the subspace defined by $\mathbf{u} = H\mathbf{w}$. Consider the smallest sphere around $\mathbf{v}$ (in the corresponding norm) that touches the subspace $\mathbf{u} = H\mathbf{w}$ (shown in each figure). The radius r_0 of this sphere is the distance of $\mathbf{v}$ from the subspace, and the tangent point $\mathbf{v}_0$ (which is not necessarily unique for L_1 projection) is the projection of $\mathbf{v}$. For this point, $\mathbf{v}_0 = H\mathbf{w}_0$ holds. The shaded region indicates the region $\{\mathbf{u} : \|\mathbf{u}\|_\infty \leq \|\mathbf{v}\|_\infty\}$. To ensure the convergence of FVI, the projected vector $\mathbf{v}_0$ must fall into the shaded region.

cone: for each $1 < i \leq N$, $|z_i| \leq z_1$ (as z_1 is the largest coordinate). Consequently, for small enough ϵ, $\mathbf{z} - \epsilon\mathbf{z} \in S(\mathbf{z}_\delta, \delta)$.

Fix such an ϵ and let $\mathbf{q} = (1-\epsilon)\mathbf{z}$. This vector is (a) contained in the image space of H because $H[(1-\epsilon)\mathbf{w}] = \mathbf{q}$; (b) $\|\mathbf{q} - \mathbf{v}\|_1 \leq \|\mathbf{z} - \mathbf{v}\|_1 = r$. The vector $\mathbf{z}$ was chosen so that it has the smallest L_1-norm in the image space of H, so the inequality cannot be sharp, i.e., $\|\mathbf{q} - \mathbf{v}\|_1 = r$. However, $\|\mathbf{q}\|_2 = (1-\epsilon)\|\mathbf{z}\|_2 < \|\mathbf{z}\|_2$ with strict inequality, which contradicts our assumption about $\mathbf{z}$, thus completing the proof. □

A.2 Probabilistic interpretation of $N(H^T)$

Definition 1. *The basis functions $\{h_k\}_{k=1}^{n_b}$ have the* uniform covering (UC) *property, if all row sums in the corresponding H matrix are identical:*

$$\sum_{k=1}^{n_b} H_{\mathbf{x},k} = B \qquad \text{for all } \mathbf{x} \in \mathbf{X},$$

and all entries are non-negative. Without loss of generality we may assume that all rows sum up to 1, i.e., H is a stochastic matrix.

We shall introduce an auxiliary MDP $\overline{\mathcal{M}}$ such that exact value iteration in $\overline{\mathcal{M}}$ is identical to the approximate value iteration in the original MDP $\mathcal{M}$. Let $\mathbf{S}$ be an K-element state space with states $\mathbf{s}_1, \ldots, \mathbf{s}_K$. A state $\mathbf{s}$ is considered a discrete observation of the true state of the system, $\mathbf{x} \in \mathbf{X}$.

The action space A and the discount factor γ are identical to the corresponding items of $\mathcal{M}$, and an arbitrary element $\mathbf{s}_0 \in \mathbf{S}$ is selected as initial state. In order to

obtain the transition probabilities, let us consider how one can get from observing s to observing s′ in the next time step: from observation s, we can infer the hidden state x of the system; in state x, the agent makes action a and transfers to state x′ according to the original MDP; after that, we can infer the probability that our observation will be s′, given the hidden state x′. Consequently, the transition probability $\overline{P}(\mathrm{s}' \mid \mathrm{s}, a)$ can be defined as the total probability of all such paths:

$$\overline{P}(\mathrm{s}' \mid \mathrm{s}, a) := \sum_{\mathrm{x},\mathrm{x}' \in \mathbf{X}} \Pr(\mathrm{x} \mid \mathrm{s}) \Pr(\mathrm{x}' \mid \mathrm{x}, a) \Pr(\mathrm{s}' \mid \mathrm{x}).$$

Here the middle term is just the transition probability in the original MDP, the rightmost term is $H_{\mathrm{x},\mathrm{s}}$, and the leftmost term can be rewritten using Bayes' law (assuming a uniform prior on x):

$$\Pr(\mathrm{x} \mid \mathrm{s}) = \frac{\Pr(\mathrm{s} \mid \mathrm{x}) \Pr(\mathrm{x})}{\sum_{\mathrm{x}'' \in \mathbf{X}} \Pr(\mathrm{s} \mid \mathrm{x}'') \Pr(\mathrm{x}'')} = \frac{H_{\mathrm{x},\mathrm{s}} \cdot \frac{1}{|\mathbf{X}|}}{\sum_{\mathrm{x}'' \in \mathbf{X}} H_{\mathrm{x}'',\mathrm{s}} \cdot \frac{1}{|\mathbf{X}|}} = \frac{H_{\mathrm{x},\mathrm{s}}}{\sum_{\mathrm{x}'' \in \mathbf{X}} H_{\mathrm{x}'',\mathrm{s}}}.$$

Consequently,

$$\overline{P}(\mathrm{s}' \mid \mathrm{s}, a) = \sum_{\mathrm{x},\mathrm{x}' \in \mathbf{X}} \frac{H_{\mathrm{x},\mathrm{s}}}{\sum_{\mathrm{x}'' \in \mathbf{X}} H_{\mathrm{x}'',\mathrm{s}}} P(\mathrm{x}' \mid \mathrm{x}, a) H_{\mathrm{x},\mathrm{s}} = \left[\mathcal{N}(H)^T P^a H\right]_{\mathrm{s},\mathrm{s}'}.$$

The rewards can be defined similarly:

$$\overline{R}(\mathrm{s}, a) := \sum_{\mathrm{x} \in \mathbf{X}} \Pr(\mathrm{x} \mid \mathrm{s}) R(\mathrm{x}, a) = \left[\mathcal{N}(H)^T \mathrm{r}^a\right]_{\mathrm{s}}.$$

It is easy to see that approximate value iteration in $\mathcal{M}$ corresponds to exact value iteration in the auxiliary MDP $\overline{\mathcal{M}}$.

A.3 The proof of the sampling theorem (theorem 1)

First we prove a useful lemma about approximating the product of two large matrices. Let $A \in \mathbb{R}^{m \times n}$ and $B \in \mathbb{R}^{n \times k}$ and let $C = A \cdot B$. Suppose that we sample columns of A uniformly at random (with repetition), and we also select the corresponding rows of B. Denote the resulting matrices with $\widehat{A}$ and $\widehat{B}$. We will show that $A \cdot B \approx c \cdot \widehat{A} \cdot \widehat{B}$, where c is a constant scaling factor compensating for the dimension decrease of the sampled matrices. The following lemma is similar to Lemma 11 of [11], but here we estimate the infinity-norm instead of the L_2-norm.

Lemma 4. *Let $A \in \mathbb{R}^{m \times N}$, $B \in \mathbb{R}^{N \times k}$ and $C = A \cdot B$. Let N' be an integer so that $1 \leq N' \leq N$, and for each $i \in \{1, \ldots, N'\}$, let r_i be an uniformly random integer from the interval $[1, N]$. Let $\widehat{A} \in \mathbb{R}^{m \times N'}$ be the matrix whose i^{th} column is the ${r_i}^{th}$ column of A, and denote by $\widehat{B}$ the $N' \times k$ matrix that is obtained by sampling the rows of B similarly. Furthermore, let*

$$\widehat{C} = \frac{N}{N'} \widehat{A} \cdot \widehat{B} = \frac{N}{N'} \sum_{i=1}^{N'} A_{*,r_i} B_{r_i,*}.$$

Then, for any $\epsilon, \delta > 0$, $\|\widehat{C} - C\|_\infty \leq \epsilon N \|A\|_\infty \|B^T\|_\infty$ *with probability at least* $1 - \delta$, *if the sample size satisfies* $N' \geq \frac{2m^2}{\epsilon^2} \log \frac{2km}{\delta}$.

Proof. We begin by bounding individual elements of the matrix $\widehat{C}$: consider the element

$$\widehat{C}_{pq} = \frac{N}{N'} \sum_{i=1}^{N'} A_{p,r_i} B_{r_i,q}.$$

Let $\mathcal{C}_{pq}$ be the discrete probability distribution determined by mass points $\{A_{p,i} \cdot B_{i,q} \mid 1 \leq i \leq N\}$. Note that $\widehat{C}_{pq}$ is essentially the sum of N' random variables drawn uniformly from distribution $\mathcal{C}_{pq}$. Clearly,

$$|A_{pi} B_{iq}| \leq \max_{ij} |A_{ij}| \max_{ij} |B_{ij}| \leq \max_i \sum_j |A_{ij}| \max_i \sum_j |B_{ij}| = \|A\|_\infty \|B\|_\infty,$$

so we can apply Hoeffding's inequality to obtain

$$\Pr\Big(\Big| \frac{\sum_{i=1}^{N'} A_{p,r_i} B_{r_i,q}}{N'} - \frac{\sum_{j=1}^{N} A_{p,j} B_{j,q}}{N} \Big| > \epsilon_1 \Big) < 2 \exp\Big(- \frac{N' \epsilon_1^2}{2 \|A\|_\infty^2 \|B\|_\infty^2} \Big),$$

or equivalently,

$$\Pr\Big(\Big| \frac{N}{N'} [\widehat{A}\widehat{B}]_{pq} - [AB]_{pq} \Big| > N\epsilon_1 \Big) < 2 \exp\Big(- \frac{N' \epsilon_1^2}{2 \|A\|_\infty^2 \|B\|_\infty^2} \Big),$$

where $\epsilon_1 > 0$ is a constant to be determined later. From this, we can bound the row sums of $\widehat{C} - C$:

$$\Pr\Big(\sum_{p=1}^{m} \big| \widehat{C}_{pq} - C_{pq} \big| > m \cdot N\epsilon_1 \Big) < 2m \exp\Big(- \frac{N' \epsilon_1^2}{2 \|A\|_\infty^2 \|B\|_\infty^2} \Big),$$

which gives a bound on $\|\widehat{C} - C\|_\infty$. This is the maximum of these row sums:

$$\begin{aligned} \Pr\Big(\|\widehat{C} - C\|_\infty > mN\epsilon_1 \Big) &= \Pr\Big(\max_q \sum_{p=1}^{m} \big| \widehat{C}_{pq} - C_{pq} \big| > mN\epsilon_1 \Big) \\ &< 2km \exp\Big(- \frac{N' \epsilon_1^2}{2 \|A\|_\infty^2 \|B\|_\infty^2} \Big). \end{aligned}$$

Therefore, by substituting $\epsilon_1 = \epsilon \|A\|_\infty \|B\|_\infty / m$, the statement of the lemma is satisfied if $2km \exp\Big(- \frac{N' \epsilon^2}{2m^2} \Big) \leq \delta$, i.e, if $N' \geq \frac{2m^2}{\epsilon^2} \log \frac{2km}{\delta}$. □

If both A and B are structured, we can sharpen the lemma to give a much better (potentially exponentially better) bound. For this, we need the following definition:

For any index set Z, a matrix A is called Z-local-scope matrix, if each column of A represents a local-scope function with scope Z.

Lemma 5. *Let A^T and B be local-scope matrices with scopes Z_1 and Z_2, and let $N_0 = n^{|Z_1|+|Z_2|}$, and apply the random row/column selection procedure of the previous lemma. Then, for any $\epsilon, \delta > 0$, $\|\widehat{C} - C\|_\infty \leq \epsilon N_0 \|A\|_\infty \|B\|_\infty$ with probability at least $1-\delta$, if the sample size satisfies $N' \geq \frac{2m^2}{\epsilon^2} \log \frac{2km}{\delta}$.*

Proof. Fix a variable assignment $\mathbf{x}[Z_1 \cup Z_2]$ on the domain $Z_1 \cup Z_2$ and consider the rows of A that correspond to a variable assignment compatible to $\mathbf{x}[Z_1 \cup Z_2]$, i.e., they are identical to it for components $Z_1 \cup Z_2$ and are arbitrary on

$$W := \{1, 2, \ldots, m\} \setminus (Z_1 \cup Z_2).$$

It is easy to see that all of these rows are identical because of the local-scope property. The same holds for the columns of B. All the equivalence classes of rows/columns have cardinality

$$N_1 := n^{|W|} = N/N_0.$$

Now let us define the $m \times N_0$ matrix A' so that only one column is kept from each equivalence class, and define the $N_0 \times k$ matrix B' similarly, by omitting rows. Clearly,

$$A \cdot B = N_1 A' \cdot B',$$

and we can apply the sampling lemma to the smaller matrices A' and B' to get that for any $\epsilon, \delta > 0$ and sample size $N' \geq \frac{2m^2}{\epsilon^2} \log \frac{2km}{\delta}$, with probability at least $1-\delta$,

$$\|\widehat{A' \cdot B'} - A' \cdot B'\|_\infty \leq \epsilon N_0 \|A'\|_\infty \|B'\|_\infty.$$

Exploiting the fact that the max-norm of a matrix is the maximum of row norms, $\|A'\|_\infty = \|A\|_\infty / N_1$ and $\|B'\|_\infty = \|B\|_\infty$, we can multiply both sides to get

$$\|N_1 \widehat{A' \cdot B'} - A \cdot B\|_\infty \leq \epsilon N_0 N_1 \|A\|_\infty / N_1 \|B\|_\infty = \epsilon N_0 \|A\|_\infty \|B\|_\infty,$$

which is the statement of the lemma. □

Note that if the scopes Z_1 and Z_2 are small, then the gain compared to the previous lemma can be exponential.

Lemma 6. *Let $A = A_1 + \ldots + A_p$ and $B = B_1 + \ldots + B_q$ where all A_i and B_j are local-scope matrices with domain size at most z, and let $N_0 = n^z$. If we apply the random row/column selection procedure, then for any $\epsilon, \delta > 0$, $\|\widehat{C} - C\|_\infty \leq \epsilon N_0 pq \max_i \|A_i\|_\infty \max_j \|B_j\|_\infty$ with probability at least $1-\delta$, if the sample size satisfies $N' \geq \frac{2m^2}{\epsilon^2} \log \frac{2km}{\delta}$.*

Proof.

$$\|\widehat{C} - C\|_\infty \leq \sum_{i=1}^{p} \sum_{j=1}^{q} \|\widehat{A_i \cdot B_j} - A_i \cdot B_j\|_\infty.$$

For each individual product term we can apply the previous lemma. Note that we can use the same row/column samples for each product, because independence is required only *within* single matrix pairs. Summing the right-hand sides gives the statement of the lemma. □

Now we can complete the proof of Theorem 1:

Proof.

$$\begin{aligned}\|\mathbf{w}^* - \mathbf{w}'\|_\infty &= \|GTH\mathbf{w}^* - \widehat{G}T\widehat{H}\mathbf{w}'\|_\infty \\ &\leq \|GTH\mathbf{w}^* - \widehat{G}T\widehat{H}\mathbf{w}^*\|_\infty + \|\widehat{G}T\widehat{H}\mathbf{w}^* - \widehat{G}T\widehat{H}\mathbf{w}'\|_\infty \\ &\leq \|GTH\mathbf{w}^* - \widehat{G}T\widehat{H}\mathbf{w}^*\|_\infty + \gamma\|\mathbf{w}^* - \mathbf{w}'\|_\infty,\end{aligned}$$

i.e., $\|\mathbf{w}^* - \mathbf{w}'\|_\infty \leq \frac{1}{1-\gamma}\|GTH\mathbf{w}^* - \widehat{G}T\widehat{H}\mathbf{w}^*\|_\infty$. Let π_0 be the greedy policy with respect to the value function $H\mathbf{w}^*$. With its help, we can rewrite $TH\mathbf{w}^*$ as a linear expression: $TH\mathbf{w}^* = \mathbf{r}^{\pi_0} + \gamma P^{\pi_0}H\mathbf{w}^*$. Furthermore, T is a componentwise operator, so we can express its effect on the downsampled value function as $T\widehat{H}\mathbf{w}^* = \widehat{\mathbf{r}}^{\pi_0} + \gamma\widehat{P^{\pi_0}H}\mathbf{w}^*$. Consequently,

$$\|GTH\mathbf{w}^* - \widehat{G}T\widehat{H}\mathbf{w}^*\|_\infty \leq \|G\mathbf{r}^{\pi_0} - \widehat{G}\widehat{\mathbf{r}}^{\pi_0}\|_\infty + \gamma\|GP^{\pi_0}H - \widehat{G}\widehat{P^{\pi_0}H}\|_\infty\|\mathbf{w}^*\|_\infty$$

Applying the previous lemma two times, we get that with probability greater than $1-\delta_1$, $\|G\mathbf{r}^{\pi_0} - \widehat{G}\widehat{\mathbf{r}}^{\pi_0}\|_\infty \leq \epsilon_1 C_1$ if $N' \geq \frac{2m^2}{\epsilon_1^2}\log\frac{2m}{\delta_1}$ and with probability greater than $1-\delta_2$, $\|GP^{\pi_0}H - \widehat{G}\widehat{P^{\pi_0}H}\|_\infty \leq \epsilon_2 C_2$ if $N' \geq \frac{2m^2}{\epsilon_2^2}\log\frac{2m^2}{\delta_2}$; where C_1 and C_2 are constants depending polynomially on N_0 and the norm of the component local-scope functions, but independent of N.

Using the notation $M = \frac{1}{1-\gamma}\Big(C_1 + \gamma C_2\|\mathbf{w}^*\|_\infty\Big)$, $\epsilon_1 = \epsilon_2 = \epsilon/M$, $\delta_1 = \delta_2 = \delta/2$ and $\Xi = M^2$ proves the theorem. □

Informally, this theorem tells that the required number of samples grows quadratically with the desired accuracy $1/\epsilon$ and logarithmically with the required certainty $1/\delta$, furthermore, the dependence on the number of variables m is slightly worse than quadratic. This means that even if the number of equations is exponentially large, i.e., $N = O(e^m)$, we can select a polynomially large random subset of the equations so that with high probability, the solution does not change very much.

References

[1] Arnborg, Stefan, Corneil, Derek G., and Proskurowski, Andrzej. Complexity of finding embeddings in a k-tree. *SIAM Journal on Algebraic and Discrete Methods*, 8(2):277–284, 1987.

[2] Baird, Leemon C. Residual algorithms: Reinforcement learning with function approximation. In *Proceedings of the International Conference on Machine Learning*, pages 30–37, 1995.

[3] Bellman, Richard E. *Adaptive Control Processes.* Princeton University Press, Princeton, NJ., 1961.

[4] Bertsekas, Dimitri P. and Tsitsiklis, John N. *Neuro-Dynamic Programming.* Athena Scientific, 1996.

[5] Boutilier, Craig, Dearden, Richard, and Goldszmidt, Moises. Exploiting structure in policy construction. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, pages 1104–1111, 1995.

[6] Boutilier, Craig, Dearden, Richard, and Goldszmidt, Moises. Stochastic dynamic programming with factored representations. *Artificial Intelligence*, 121(1-2):49–107, 2000.

[7] de Farias, Daniela Pucci and van Roy, Benjamin. Approximate dynamic programming via linear programming. In *Advances in Neural Information Processing Systems 14*, pages 689–695, 2001.

[8] de Farias, Daniela Pucci and van Roy, Benjamin. On constraint sampling in the linear programming approach to approximate dynamic programming. *Mathematics of Operations Research*, 29(3):462–478, 2004.

[9] Dechter, Rina. Bucket elimination: A unifying framework for reasoning. *Artificial Intelligence*, 113(1-2):41–85, 1999.

[10] Dolgov, Dmitri A. and Durfee, Edmund H. Symmetric primal-dual approximate linear programming for factored MDPs. In *Proceedings of the 9th International Symposium on Artificial Intelligence and Mathematics (AI&M 2006)*, 2006.

[11] Drineas, Petros, Kannan, Ravi, and Mahoney, Michael W. Fast monte carlo algorithms for matrices i: Approximating matrix multiplication. *SIAM Journal of Computing*, 36:132–157, 2006.

[12] Drineas, Petros, Mahoney, Michael W., and Muthukrishnan, S. Sampling algorithms for l2 regression and applications. In *Proc. 17-th Annual SODA*, pages 1127–1136, 2006.

[13] Guestrin, Carlos, Koller, Daphne, Gearhart, Chris, and Kanodia, Neal. Generalizing plans to new environments in relational MDPs. In *Eighteenth International Joint Conference on Artificial Intelligence*, 2003.

[14] Guestrin, Carlos, Koller, Daphne, Parr, Ronald, and Venkataraman, Shobha. Efficient solution algorithms for factored MDPs. *Journal of Artificial Intelligence Research*, 19:399–468, 2002.

[15] Guestrin, Carlos, Patrascu, Relu, and Schuurmans, Dale. Algorithm-directed exploration for model-based reinforcement learning in factored mdps. In *Proceedings of the International Conference on Machine Learning*, pages 235–242, 2002.

[16] Kearns, Michael J. and Koller, Daphne. Efficient reinforcement learning in factored MDPs. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence*, pages 740–747, 1999.

[17] Koller, Daphne and Parr, Ronald. Policy iteration for factored mdps. In *Proceedings of the 16th Conference on Uncertainty in Artificial Intelligence*, pages 326–334, 2000.

[18] Kveton, Branislav, Hauskrecht, Milos, and Guestrin, Carlos. Solving factored MDPs with hybrid state and action variables. *Journal of Artificial Intelligence Research*, 27:153–201, 2006.

[19] Lagoudakis, Michail G. and Parr, Ronald. Least-squares policy iteration. *Journal of Machine Learning Research*, 4:1107–1149, 2003.

[20] Liberatore, Paolo. The size of MDP factored policies. In *Proceedings of the 18th National Conference on Artificial intelligence*, pages 267–272, 2002.

[21] Patrascu, Relu, Poupart, Pascal, Schuurmans, Dale, Boutilier, Craig, and Guestrin, Carlos. Greedy linear value-approximation for factored markov decision processes. In *Proceedings of the 18th National Conference on Artificial intelligence*, pages 285–291, 2002.

[22] Poupart, Pascal, Boutilier, Craig, Patrascu, Relu, and Schuurmans, Dale. Piecewise linear value function approximation for factored mdps. In *Proceedings of the 18th National Conference on AI*, 2002.

[23] Sallans, Brian. *Reinforcement Learning for Factored Markov Decision Processes*. PhD thesis, University of Toronto, 2002.

[24] Sanner, Scott and Boutilier, Craig. Approximate linear programming for first-order MDPs. In *Proceedings of the 21th Annual Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 509–517, 2005.

[25] Schweitzer, Paul J. and Seidmann, Abraham. Generalized polynomial approximations in markovian decision processes. *Journal of Mathematical Analysis and Applications*, 110(6):568–582, 1985.

[26] Sutton, Richard S. and Barto, Andrew G. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, 1998.

[27] Tsitsiklis, John N. and Roy, Benjamin Van. An analysis of temporal-difference learning with function approximation. *IEEE Transactions on Automatic Control*, 42(5):674–690, 1997.

Acta Cybernetica 18 (2008) 637–649.

InfoMax Bayesian Learning of the Furuta Pendulum

László A. Jeni*, György Flórea† and András Lőrincz†‡

Abstract

We have studied the InfoMax (D-optimality) learning for the two-link Furuta pendulum. We compared InfoMax and random learning methods. The InfoMax learning method won by a large margin, it visited a larger domain and provided better approximation during the same time interval. The advantages and the limitations of the InfoMax solution are treated.

Keywords: Online Bayesian learning, D-optimality, infomax control, Furuta pendulum

1 Introduction

In recent years, machine learning methods became more and more accurate and popular, so that the task of learning the dynamics of plants and the learning of reactive behaviours to environmental changes seem to be within reach. Such tasks call for online (i.e., real time) learning methods. For fast learning, one would like to provide stimuli that facilitate the fastest information gain about the changes of the plant and its environment [3, 2].

As an example, consider an industrial robot. Programming of industrial robots is traditionally an off-line task. In typical situations, the trajectory of the robot is generated from a CAD model of the environment and the robot [12]. However, this model holds no information about the unavoidable modelling errors especially if the environment changes. We assume that the environment and the robot have a parametrised representation and that the goal is to estimate these parameters as quickly as possible and possibly on-the-flight.

An attractive route replaces trajectory planning and trajectory tracking with speed-field planning and speed-field tracking. This latter is less strict about the actual path. The difference between the two methods can be described by the example when one is walking on a crowded street. Here, the trajectory should be

*Eötvös Loránd University, Department of Software Technology and Methodology, E-mail: jedi@inf.elte.hu

†Eötvös Loránd University, Department of Information Systems, E-mail: ripps11@freemail.hu, andras.lorincz@elte.hu

‡Corresponding author

replanned at each time instant and at full length when anybody moves. Speed-field, however, undergoes slight changes even if the walker is pushed by the crowd. Further, speed-field tracking requires a crude model of the inverse dynamics and still has attractive global stability properties [13, 14].

This underlines our approach, where we try to approximate the dynamics fast. We use the InfoMax (also called D-optimality) approach. InfoMax learning means that the next control action is chosen according to Bayesian estimation given what we have learnt until *now*, given the actual state that we have *at this moment*. The task is the estimation of the best control signal *now* to gain the most information about the unknown parameters of the model from the next observation. Note, however, that the state of the unknown plant is also an issue, we may not know the order of the dynamics, or the temporal convolution corrupting instantaneous control actions. We treat the problem of the order of the dynamics here.

It has been shown recently by Póczos and Lőrincz [11] that InfoMax control can be computed analytically without approximations and leads to simple learning rules by slightly modifying the generalised linear model in [8].

Our particular example that we study is the so called *two-link Furuta pendulum* [4]. This pendulum as well as the related InfoMax task are sketched in Fig. 1.

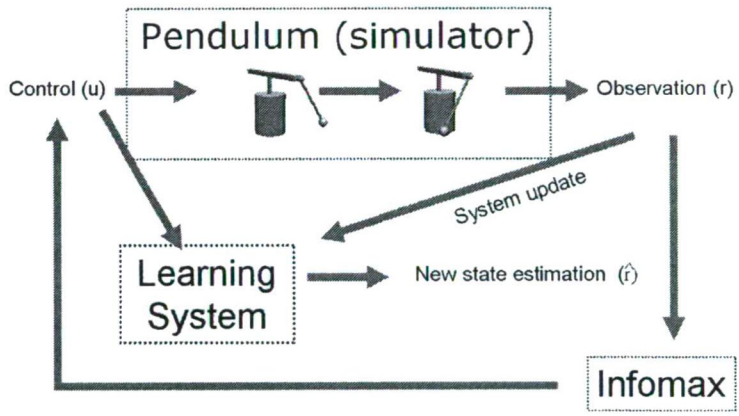


Figure 1: Furuta pendulum and InfoMax learning.
The pendulum stimulator receives the control input from the InfoMax algorithm, the only output of this exploratory algorithm. The learning system also receives this control signal. The learning system estimates, the pendulum stimulator computes the next state. All information, i.e., state, state estimation, control signal are used to update the parameters of the learning system and to compute the next control signal by the InfoMax algorithm.

The paper is organised as follows. In Section 2, we review the InfoMax approach. Section 3 is about the dynamics and the parametrisation of the Furuta pendulum problem. Section 4 describes the results. We summarise our findings and draw conclusions in Section 5.

2 Infomax Learning

We introduce the model used in [11]. Let us assume that we have d simple computational units called '*neurons*' in a recurrent neural network:

$$r_{t+1} = g\left(\sum_{i=0}^{I} F_i r_{t-i} + \sum_{j=0}^{J} B_j u_{t+1-j} + e_{t+1}\right), \quad (1)$$

where $\{e_t\}$, the driving noise of the RNN, denotes temporally independent and identically distributed (i.i.d.) stochastic variables and $P(e_t) = \mathcal{N}_{e_t}(0, V)$, $r_t \in \mathcal{R}^d$ represents the observed activities of the neurons at time t. Let $u_t \in \mathbb{R}^c$ denote the control signal at time t. The neural network is formed by the weighted delays represented by matrices F_i $(i = 0, \ldots, I)$ and B_j $(j = 0, \ldots, J)$, which connect neurons to each other and also the control components to the neurons, respectively. Control can also be seen as the means of interrogation, or the stimulus to the network [8]. We assume that function $g : \mathbb{R}^d \to \mathbb{R}^d$ in (1) is known and invertible. The computational units, the neurons, sum up weighted previous neural activities as well as weighted control inputs. These sums are then passed through identical non-linearities according to (1). The goal is to estimate the parameters $F_i \in \mathbb{R}^{d\times d}$ $(i = 0, \ldots, I)$, $B_j \in \mathbb{R}^{d\times c}$ $(j = 0, \ldots, J)$ and the covariance matrix V, as well as the driving noise e_t by means of the control signals.

We introduce the following notations:

$$x_{t+1} = [r_{t-I}; \ldots; r_t; u_{t-J+1}; \ldots; u_{t+1}], \quad (2)$$

$$y_{t+1} = g^{-1}(r_{t+1}), \quad (3)$$

$$A = [F_I, \ldots, F_0, B_J, \ldots, B_0] \in \mathbb{R}^{d\times m}. \quad (4)$$

Using these notations, the original model (1) reduces to a linear equation:

$$y_t = Ax_t + e_t. \quad (5)$$

The InfoMax learning relies on Bayes' method in the online estimation of the unknown quantities (parameter matrix A, noise e_t and its covariance matrix V). It assumes that prior knowledge is available and it updates the posteriori knowledge on the basis of the observations. Control will be chosen at each instant to provide maximal expected information concerning the quantities we have to estimate. Starting from an arbitrary prior distribution of the parameters the posterior distribution needs to be computed. This estimation can be highly complex, so approximations are common in the literature. For example, assumed density filtering, when the computed posterior is projected to simpler distributions, has been suggested [1, 10, 9]. Póczos and Lőrincz [11] used the method of conjugated priors [6], instead. For matrix A we assume a matrix valued normal (i.e., Gaussian) distribution prior. For covariance matrix V inverted Wishart (IW) [7] distribution

will be our prior. There are three advantages of this choice: (i) they are somewhat more general than the typical Gaussian assumption, (ii) the functional form of the posteriori distributions is not affected, and (iii) the model (2-4) admits analytical – i.e., approximation-free – solution for this case as shown in [11]. Below, we review the main concepts and the crucial steps of this analytical solution.

Let us define the normally distributed matrix valued stochastic variable $A \in \mathbb{R}^{d\times m}$ by using the following quantities: $M \in \mathbb{R}^{d\times m}$ is the expected value of A. $V \in \mathbb{R}^{d\times d}$ is the covariance matrix of the rows, and $K \in \mathbb{R}^{m\times m}$ is the so-called precision parameter matrix that we shall modify in accordance with the Bayesian update. They are both positive semi-definite matrices.

Now, one can rewrite model (5) as follows:

$$\begin{aligned} P(A|V) &= \mathcal{N}_A(M,V,K), && (6)\\ P(V) &= \mathcal{IW}_V(Q,n), && (7)\\ P(e_t|V) &= \mathcal{N}_{e_t}(0,V), && (8)\\ P(y_t|A,x_t,V) &= \mathcal{N}_{y_t}(Ax_t,V). && (9) \end{aligned}$$

We introduce the following quantities:

$$\begin{aligned} \gamma_{t+1} &= 1 - x_{t+1}^T(x_{t+1}x_{t+1}^T + K_t)^{-1}x_{t+1},\\ n_{t+1} &= n_t + 1,\\ M_{t+1} &= (M_tK_t + y_{t+1}x_{t+1}^T)(x_{t+1}x_{t+1}^T + K_t)^{-1},\\ Q_{t+1} &= Q_t + (y_{t+1} - M_tx_{t+1})\,\gamma_{t+1}\,(y_{t+1} - M_tx_{t+1})^T, \end{aligned} \quad (10)$$

for the posterior probabilities. Then – one can show [11] that –

$$\begin{aligned} P(A|V,\{x\}_1^{t+1},\{y\}_1^{t+1}) &= \mathcal{N}_A(M_{t+1},V,x_{t+1}x_{t+1}^T + K_t), && (11)\\ P(V|\{x\}_1^{t+1},\{y\}_1^{t+1}) &= \mathcal{IW}_V\left(Q_{t+1},n_{t+1}\right), && (12)\\ P(y_{t+1}|\{x\}_1^{t+1},\{y\}_1^{t}) &= \mathcal{T}_{y_{t+1}}\left(Q_t,n_t,M_tx_{t+1},\gamma_{t+1}\right). \end{aligned}$$

The derivations give rise to a strikingly simple optimal control value expression:

$$u_{t+1^{opt}} = \arg\max_{u\in\mathcal{U}} x_{t+1}^T K_t^{-1} x_{t+1}, \quad (13)$$

The steps of the InfoMax update are summarised in Algorithm 1. We shall follow these steps in our computer studies on the Furuta pendulum that we detail in the next section.

3 The Furuta pendulum

3.1 Furuta Pendulum

The Furuta pendulum is shown in Figure 2. The pendulum has two links [4, 5]. Configuration of the pendulum is determined by the length of the links and by two

Algorithm 1 Pseudocode of the InfoMax algorithm.

Control Calculation

1: $u_{t+1} = \arg\max_{u \in \mathcal{U}} \hat{x}_{t+1}^T K_t^{-1} \hat{x}_{t+1}$
2: where $\hat{x}_{t+1} = [r_{t-I}; \ldots; r_t; u_{t-J+1}; \ldots; u_t; u]$
3: set $x_{t+1} = [r_{t-I}; \ldots; r_t; u_{t-J+1}; \ldots; u_t; u_{t+1}]$

Observation

1: observe r_{t+1}, and let $y_{t+1} = g^{-1}(r_{t+1})$

Bayesian update

1: $M_{t+1} = (M_t K_t + y_{t+1} x_{t+1}^T)(x_{t+1} x_{t+1}^T + K_t)^{-1}$
2: $K_{t+1} = x_{t+1} x_{t+1}^T + K_t$
3: $n_{t+1} = n_t + 1$
4: $\gamma_{t+1} = 1 - x_{t+1}^T (x_{t+1} x_{t+1}^T + K_t)^{-1} x_{t+1}$
5: $Q_{t+1} = Q_t + (y_{t+1} - M_t x_{t+1})\, \gamma_{t+1}\, (y_{t+1} - M_t x_{t+1})^T$

angles. Dynamics of the pendulum are also determined by the different masses, i.e. the masses of the links and the mass of the end effector as well as by the two actuators, which are able to rotate the horizontal link and the swinging link in both directions, respectively. The angle of the horizontal link is denoted by ϕ, whereas the symbol for the angle of the horizontal link is θ (Fig. 2). Parameters of our computer studies are provided in Table 1. The state of the pendulum is given by ϕ, θ, $\dot{\phi}$ and $\dot{\theta}$. The magnitude of the angular speeds $\dot{\phi}$ and $\dot{\theta}$ was restricted to 2 rotations/s, i.e. to the interval $[-2\frac{\text{rot}}{\text{s}}, 2\frac{\text{rot}}{\text{s}}]$.

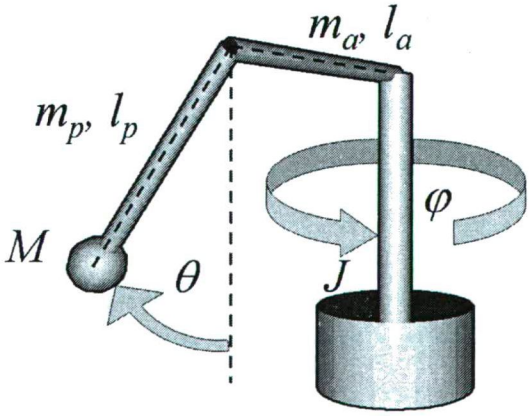


Figure 2: Furuta pendulum and notations of the different parameters.

Let τ_ϕ and τ_θ denote the external torques applied to the vertical arm and to the horizontal arm, respectively. Introducing

Name of parameter	Value	Unit	Notation
Angle of swinging link		rad	θ
Angle of horizontal link		rad	ϕ
Mass of horizontal link	0.072	kg	m_a
Mass of vertical link	0.00775	kg	m_p
Mass of the weight	0.02025	kg	M
Length of horizontal link	0.25	m	l_a
Length of vertical link	0.4125	m	l_p
Coulomb friction	0.015	Nm	τ_S
Coulomb stiction	0.01	Nm	τ_C
Maximal rotation speed for both links	2	$\frac{\text{rotation}}{s}$	
Approx. zero angular speed for swinging link	0.02	$\frac{rad}{s}$	$\dot{\phi}_\epsilon$
Time intervals between interrogations	100	ms	
Maximum control value	0.05	Nm	δ

Table 1: Parameters of the Physical Model

$$\alpha = J + (M + \frac{1}{3}m_a + m_p)l_a^2, \tag{14}$$

$$\beta = (M + \frac{1}{3}m_p)l_p^2, \tag{15}$$

$$\gamma = (M + \frac{1}{2}m_p)l_a l_p, \tag{16}$$

$$\delta = (M + \frac{1}{2}m_p)g l_p \tag{17}$$

and using the external torques, we can write the equations of the dynamics of the pendulum as follows [5]:

$$(\alpha + \beta sin^2\theta)\ddot{\phi} + \gamma cos\theta\,\ddot{\theta} + 2\beta cos\theta\, sin\theta\,\dot{\phi}\,\dot{\theta} - \gamma sin\theta\,\dot{\theta}^2 = \tau_\phi \tag{18}$$

$$\gamma cos\theta\,\ddot{\phi} + \beta\ddot{\theta} - \beta cos\theta\, sin\theta\,\dot{\phi}^2 - \delta sin\theta = \tau_\theta \tag{19}$$

The real pendulum exhibits significant friction in the $\phi-$joint. The friction can be modelled in several ways. We used Coulomb friction with stiction [5]:

$$\tau_F = \begin{cases} \tau_C\, sgn\dot{\phi} & \text{if } \dot{\phi} \neq 0, \\ \tau_u & \text{if } \dot{\phi} = 0 \text{ and } \|\tau_u\| \leq \tau_S, \\ \tau_S\, sgn\dot{\tau}_u & \text{otherwise} \end{cases} \tag{20}$$

In our simulations the zero condition on the velocity is replaced by $\|\dot{\phi}\| \leq \dot{\phi}_\epsilon$, with $\dot{\phi}_\epsilon$ chosen according to [5].

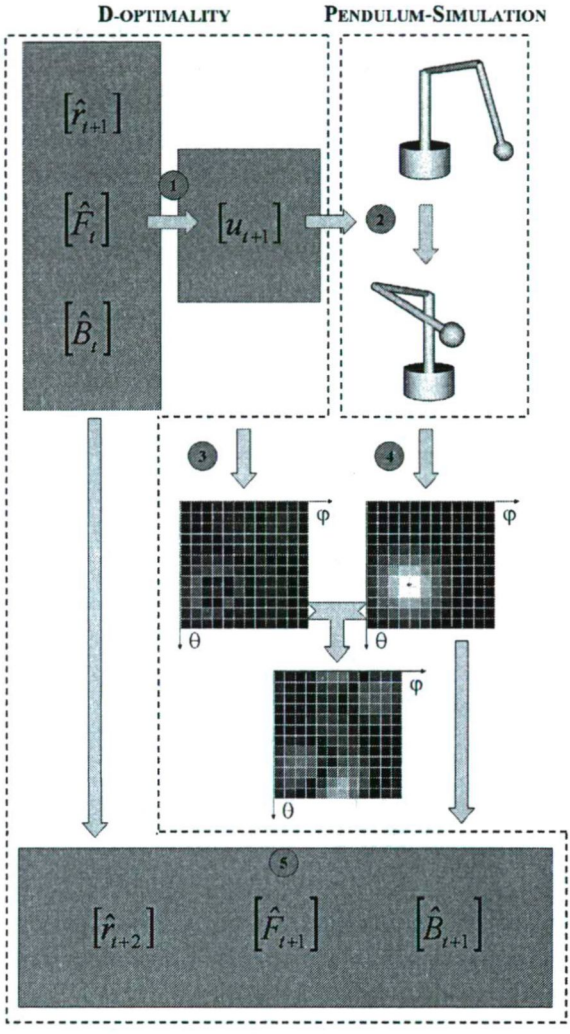


Figure 3: Scheme of D-optimal interrogation. (1) Control u_{t+1} is computed from D-optimal principle, (2) control acts upon the pendulum, (3) signals predicted before control step, (4) sensory information after control step. Difference between (3) and (4) is used for the computation of the cumulated prediction error. (5) Parameters were updated according to Algorithm 1. For more details, see text.

3.2 Simulation and learning

The pendulum is a continuous dynamical system that we observe in discrete time steps. Furthermore, we assume that our observations are limited; we have only 144

crude sensors for observing angles ϕ and θ. In each time step these sensors form our $r_t \in \mathbb{R}^{144}$ observations, which were simulated as follows: Space of angles ϕ and θ is $[0, 2\pi) \times [0, 2\pi)$. We divided this space into $12 \times 12 = 144$ squared domains of equal sizes. We 'put' a Gaussian sensor at the centre of each domain. Each sensor gives maximal response 1 when angles θ and ϕ of the pendulum are in the centre of the respective sensor, whereas the response decreased according to the Gaussian function. For all sensors, response r_i scaled as $r_i = \frac{1}{2\pi\sigma^2} \exp(-\frac{(\theta-\theta_i)^2+(\phi-\phi_i)^2}{2\sigma^2})$ $(1 \leq i \leq 144)$, where angles θ_i, ϕ_i correspond to the middle point of our 12×12 grid, and σ was set to 1.58 in radians. Sensors were crude but noise-free; no noise was added to the sensory outputs. The inset at label 4 of Figure 3 shows the outputs of the sensors in a typical case. Sensors satisfied periodic boundary conditions; if sensor S was centred around zero degree in any of the directions, then it sensed both small (around 0 radian) and large (around 2π radian) angles. We note that the outputs of the 144 domains are arranged for purposes of visualisation; the underlying topography of the sensors is hidden for the learning algorithm.

We observed these $r_t = (r_1(t), \ldots, r_i(t), \ldots, r_{144}(t))^T$ quantities and then calculated the $u_{t+1} \in \mathbb{R}^2$ D-optimal control using Algorithm 1, where we approximated the pendulum with the model $\tilde{r}_{t+1} = Fr_t + Bu_{t+1}$, $F \in \mathbb{R}^{144\times144}$, $B \in \mathbb{R}^{144\times2}$. Components of vector u_{t+1} controlled the two actuators of the angles separately. Maximal magnitude of each control signal was set to 0.05 Nm. Clearly we do not know the best parameters for F and B in this case, so in the performance measure we have to rely on prediction error and the number of visited domains. This procedure is detailed below.

First, we note that the angle of the swinging link and the angular speeds are important from the point of view of the prediction of the dynamics, whereas the angle of the horizontal link can be neglected. Thus, for the investigation of performance of the learning process, we used the 3D space determined by $\dot{\phi}, \theta$ and $\dot{\theta}$. As was mentioned above, angular speeds were restricted to the $[-2\frac{\text{rot}}{\text{s}}, 2\frac{\text{rot}}{\text{s}}]$ domain. We divided each angular speed domain into 12 equal regions. We also used the 12-fold division of angle θ. Counting the domains, we had $12 \times 12 \times 12 = 1,728$ rectangular block shaped domains. Our algorithm provides estimations for $\hat{F}_t$ and $\hat{B}_t$ in each instant. We can use them to compute the predicted observation vector $\hat{r}_{t+1} = \hat{F}_t r_t + \hat{B}_t u_{t+1}$. An example is shown in the inset with label 4 in Figure 3. We computed the absolute value of the prediction errors $e_i(t+1) = \|r_{i,t+1} - \hat{r}_{i,t+1}\|$ for all i, and cumulated them over all domains $(i = 1, \ldots 1,728)$ as follows. For each domain, we set the initial error value at 30, a value somewhat larger than the maximal error we found in the computer runs. Therefore the cumulated error at start was $1,728 \times 30 = 51,840$ and we measured how the error decreases.

4 Results

The D-optimal algorithm does two things simultaneously: (i) it explores new domains, and (ii) it decreases the errors in the domains already visited. Thus, we measured the cumulated prediction errors during learning and corrected the estimation

at each step. So, if our cumulated error estimation at time t was $e(t) = \sum_{k=1}^{1,728} e_k(t)$ and the pendulum entered the i^{th} domain at time $t+1$, then we set $e_k(t+1) = e_k(t)$ for all $k \neq i$ and $e_i(t+1) = \|r_{i,t+1} - \hat{r}_{i,t+1}\|$. Then we computed the new cumulated prediction error, i.e., $e(t+1) = \sum_{i=1}^{1,728} e_i(t+1)$.

We compared the random and the D-optimality interrogation schemes. We show two sets of figures, Figures 4a and 4b, as well as Figures 4c and 4d. The upper set depicts the results for the full set of the 1,728 domains. It is hard for the random control to enter the upper domain by chance, so we also investigated how the D-optimal control performs here. We computed the performance for cases when the swinging link was above vertical, that is for 864 domains (Figs. 4c and 4d).

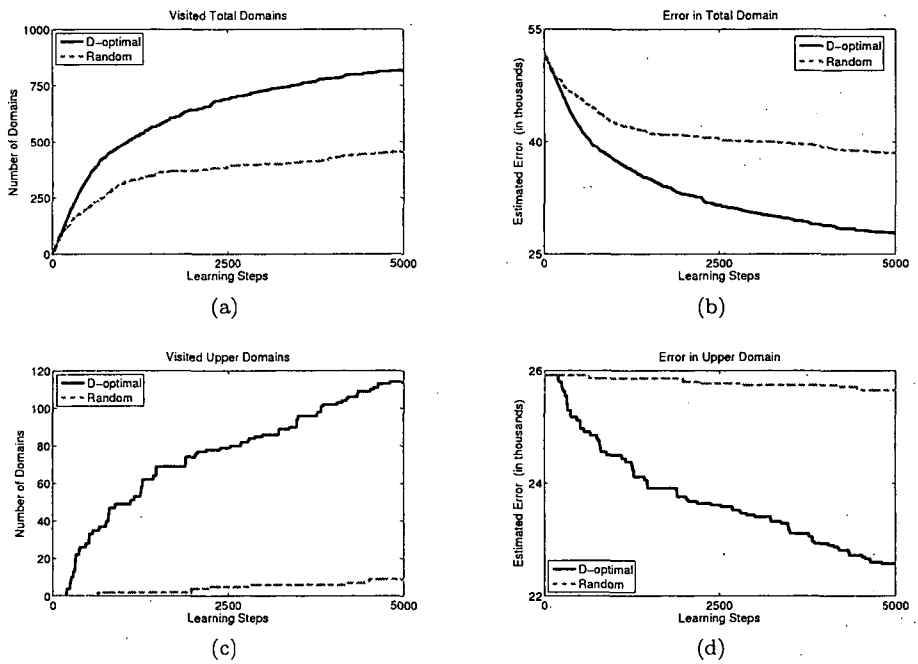


Figure 4: Furuta experiments driven by random and D-optimality controls. Solid (dotted) line: D-optimal (random) case. (a-b): Number of domains is 1728. (a): visited domains, (b): upper bound for cumulated estimation error in all domains, (c-d): Upper half of the space, i.e., the swinging angle is above horizontal and the number of domains is 864. (c): number of visited domains, (d): upper bound for cumulated estimation error. For more details, see text.

For the full space, the number of visited domains is 456 (26%) and 818 (47%) for the random control and the D-optimal control, respectively after 5,000 control steps (Fig. 4a). The error drops by 13,390 (26%) and by 24,040 (46%), respectively (Fig. 4b). While the D-optimal controlled pendulum visited more domains and

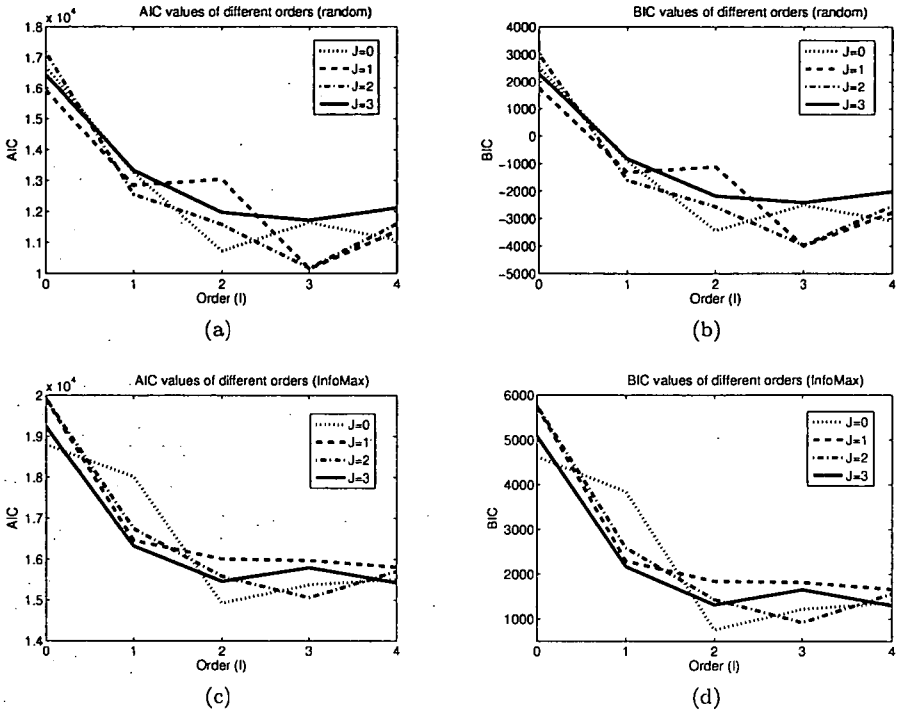


Figure 5: The Akaike's information criterion (AIC) and the Bayesian information criterion (BIC) values for random and InfoMax controls and for models up to fourth order ($I = 0, \ldots, 3$) and for different control orders ($J = 0, \ldots, 3$).

achieved smaller errors, the domain-wise estimation error is about the same for the domains visited; both methods gained about 29.4 points per domains.

We can compute the same quantities for the upper domains as well. The number of visited upper domains is 9 and 114 for the random control and for the D-optimal control, respectively (Figure 4c). The decrease of error is 265 and 3,342, respectively (Figure 4d). In other words, D-optimal control gained 29.3 points in each domain on average, whereas random control, on average, gained 29.4 points, which are very close to the previous values in both cases. That is, infomax control gains more information concerning the system to be identified by visiting new domains.

This observation is further emphasised by the following data: The infomax algorithm discovered 37 new domains in the last 500 steps of the 5,000 step experiment. Out of these 37 domains, 20 (17) were discovered in the lower (upper) domain. By contrast, the random algorithm discovered 9 domains, out which 5 (4) was in the lower (upper) domain. That is, infomax has a similar (roughly fourfold) lead in both the upper and lower domains, although the complexity of the task is different and the relative number of available volumes is also different in these two domains.

We studied the learning process as a function of the Gaussian width. We found

that learning is robust in this respect: the estimation error was a very weak function of σ, the spread of the Gaussian, except for very small variance Gaussians. Learning was spoiled for very broad Gaussians, too.

By construction, there is a second order dynamical system in the background, so we studied if one can find this order as the result of the learning process. We calculated Akaike's information criterion (AIC) and the Bayesian information criterion (BIC) values of the model for different control orders. Figure 5 shows the AIC and BIC values of both random control and for InfoMax control. We measured the values for models up to fourth order ($I = 0, \ldots, 3$), for control orders $J = 1, \ldots, 3$, and for $\sigma = 1.58$ radian. There is a large gain if one increases $I = 0$ to $I = 1$, i.e., if one assumes second order dynamics. Further increases of I give smaller improvements. The only exception is the case of $J = 0$. Here, the improvement is not so sudden between $I = 0$ and $I = 1$ and considerable further drops can be seen for $I = 2$. This is most prominent under the InfoMax conditions. Thus, there is a memory effect in the control: control rendered by InfoMax at time t may depend on the control rendered by InfoMax at time $t-1$. This dependency is learned and is represented by matrix F_2. From the point of view of the order of the control, there is little dependence here, except for the case of $I = 1$: there is a large performance difference – for InfoMax control – between $J = 0$ and $J = 1$. Again, this points to the memory effect in InfoMax, which can be uncovered by matrix B_1. Taken together, the approach can provide an *estimation about the order of the dynamical system*, but *not* in the InfoMax operation mode.

Finally, we note that the InfoMax procedure, which we demonstrated here on the case of the Furuta pendulum, may gain from discovering the direct product space behind the 144 sensors. Then explorations might concern the low-dimensional direct product space, instead of the raw sensory observations.

5 Summary and conclusion

In this paper we have studied the InfoMax learning for the two-link Furuta pendulum. We used a slightly modified version of the generalised linear model described in [8]. The intriguing property of this slight modification is that it leads to strikingly simple learning rules [11].

InfoMax intends to optimise the next control action given what has been learned and what has been observed. We demonstrated that this online (i.e., real time) learning method explores larger areas than random control without significant compromise in the precision of the estimation in the visited domains. The discovery rate is in favour of the InfoMax algorithm, which had similar leads in the domains which were easier to find and in the domains, which were harder to find.

The pendulum problem also shows the limitations of the InfoMax solution. This is a low-dimensional problem and InfoMax cannot learn the hidden regularities. Connections to reinforcement learning should be established for efficient learning. Convergent methods that can connect InfoMax learning and reinforcement learning seem important for machine learning.

References

[1] Boyen, X. and Koller, D. Tractable inference for complex stochastic processes. In *Fourteenth Conference on Uncertainty in Artificial Intelligence*, pages 33–42, 1998.

[2] Cohn, D. A. Neural network exploration using optimal experiment design. In *Advances in Neural Information Processing Systems*, volume 6, pages 679–686, 1994.

[3] Fedorov, V. V. *Theory of Optimal Experiments.* Academic Press, New York, 1972.

[4] Furuta, K., Yamakita, M., and Kobayashi, S. Swing-up control of inverted pendulum using pseudo-state feedback. *Journal of Systems and Control Engineering*, 206:263–269., 1992.

[5] Gäfvert, M. Modelling the furuta pendulum. Technical report ISRN LUTFD2/TFRT–7574–SE, Department of Automatic Control, Lund University, Sweden, April 1998.

[6] Gelman, A., Carlin, J. B., Stern, H. S., and Rubin, D. B. *Bayesian Data Analysis.* CRC Press, 2nd erdition, 2003.

[7] Gupta, A. K. and Nagar, D. K. *Matrix Variate Distributions*, volume 104 of *Monographs and Surveys in Pure and Applied Mathematics.* Chapman and Hall/CRC, 1999.

[8] Lewi, J., Butera, R., and Paninski, L. Real-time adaptive information-theoretic optimization of neurophysiology experiments. In *Advances in Neural Information Processing Systems*, volume 19, 2007.

[9] Minka, T. *A family of algorithms for approximate Bayesian inference.* PhD thesis, MIT Media Lab, MIT, 2001.

[10] Opper, M. and Winther, O. A Bayesian approach to online learning. In *Online Learning in Neural Networks.* Cambridge University Press, 1999.

[11] Póczos, B. and Lőrincz, A. D-optimal Bayesian interrogation for parameter estimation and noise identification of recurrent neural networks. Technical Report, Janury 2008. `http://uk.arxiv.org/PS_cache/arxiv/pdf/0801/0801.1883v1.pdf`.

[12] Solvang, B., Korondi, P., Sziebig, G, and Ando, N. SAPIR: Supervised and adaptive programming of industrial robots. In *11th IEEE International Conference on Intelligent Engineering Systems*, INES07, Budapest, Hungary, June 2007.

[13] Szepesvári, Cs., Cimmer, Sz., and Lőrincz, A. Neurocontroller using dynamic state feedback for compensatory control. *Neural Networks*, 10:1691–1708, 1997.

[14] Szepesvári, Cs. and Lőrincz, A. Approximate inverse-dynamics based robust control using static and dynamic feedback. *Applications of Neural Adaptive Control Theory II*, 2:151–179, 1997. World Scientific, Singapore.

Acta Cybernetica 18 (2008) 651–668.

Factored Temporal Difference Learning in the New Ties Environment*

Viktor Gyenes†, Ákos Bontovics†, and András Lőrincz†‡

Abstract

Although reinforcement learning is a popular method for training an agent for decision making based on rewards, well studied tabular methods are not applicable for large, realistic problems. In this paper, we experiment with a factored version of temporal difference learning, which boils down to a linear function approximation scheme utilising natural features coming from the structure of the task. We conducted experiments in the New Ties environment, which is a novel platform for multi-agent simulations. We show that learning utilising a factored representation is effective even in large state spaces, furthermore it outperforms tabular methods even in smaller problems both in learning speed and stability, because of its generalisation capabilities.

Keywords: reinforcement learning, temporal difference, factored MDP

1 Introduction

Reinforcement learning (RL) [16] is a framework for training an agent for a given task based on positive or negative feedback called *immediate rewards* that the agent receives in response to its actions. Mathematically, the behaviour of the agent is characterised by a *Markov decision process* (MDP), which involves the *states* the agent can be in, *actions* the agent can execute depending on the state, a *state transition model*, and the *rewards* the agent receives.

For small, discrete state spaces well-studied tabular methods exist for solving the learning task. However, real world tasks include many variables, often continuous ones, for which the state space is very large, or even infinite, making these

*This material is based upon work supported partially by the European Office of Aerospace Research and Development, Air Force Office of Scientific Research, Air Force Research Laboratory, under Contract No. FA-073029. This research has also been supported by an EC FET grant, the 'New Ties project' under contract 003752. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the European Office of Aerospace Research and Development, Air Force Office of Scientific Research, Air Force Research Laboratory, the EC, or other members of the EC New Ties project.

†Eötvös Loránd University, Department of Information Systems, E-mail: gyenesvi@inf.elte.hu, bontovic@elte.hu, andras.lorincz@elte.hu

‡Corresponding author

approaches inapplicable. Consider a sequential decision problem with m variables. In general, we need an exponentially large state space to model it as an MDP. Thus, no matter if the solution time is polynomial in the number of states, scaling is *exponential* in the size of the task description (m). Fortunately, by exploiting state space structure, it is possible to drastically reduce the time needed for solving large MDPs. It is often the case that the transition model for a given variable depends only on a few other variables. Also, rewards may be associated with only a small number of variables. In such situations, *factored* MDPs (fMDPs) offer a more compact representation by taking the state space as the Cartesian product of m variables: $X = X_1 \times X_2 \times \ldots \times X_m$. This gives rise to learning in a factored manner, by separating variables as much as possible, taking into account (a priori known) dependencies among them and circumventing combinatorial explosion.

One popular method for solving MDPs is based on *value functions* that represent long term utilities that can be collected starting from a given state. The agent's behaviour, also called *policy*, is then defined by acting greedily according to this value function, i.e. selecting actions that result in next states with highest possible value. For factored problems one may be able to define the value function as the sum of *local scope functions*, i.e. functions that depend only on a small number of variables. This way, both the memory capacity needed to store the value function, and the learning time might be reduced by an exponential factor.

Many algorithms exist for solving factored MDPs. For the *value iteration* method convergence proof for factored MDPs has been provided recently [17]. In this paper, we experiment with the more flexible *temporal difference learning* (TD) method, in which the agent updates state values it actually observes during interacting with the world, as opposed to value iteration, which iteratively updates all state values synchronously. The advantage of our method is known; value iteration requires a model, whereas no model is required for TD learning.

The rest of the paper is structured as follows: in Section 2 we review the basics of reinforcement learning, especially value iteration and TD learning, motivate factored MDPs and derive our factored TD learning algorithm. Section 3 provides an overview of other approaches to learning in factored MDPs. In Section 4 we show simulation results utilising the newly introduced factored TD learning algorithm. Section 5 discusses some decisions in the choice of our agent architecture and experimental setup, and finally Section 6 concludes the paper.

2 Factored Reinforcement Learning

2.1 Value-function based Reinforcement Learning

Consider an MDP characterised by the tuple (X, A, P, R, γ), where X is the (finite) set of states the agent can be in, A is the (finite) set of actions the agent can execute, $P : X \times A \times X \to [0, 1]$ is the transition probability model, i.e. $P(x, a, x')$ is the probability that the agent arrives at state x' when executing action a in state x, $R : X \times A \to \mathbb{R}$ is the reward function, and γ is the discount rate on future

rewards.

A (Markov) policy of the agent is a mapping $\pi : X \times A \rightarrow [0,1]$ so that $\pi(x,a)$ tells the probability that the agent chooses action a in state x. For any $x_0 \in X$, the policy of the agent determines a stochastic process experienced through the instantiation

$$x_0, a_0, r_0, x_1, a_1, r_1, \ldots, x_t, a_t, r_t, \ldots$$

where r_i is the reward received after executing action a in state x_i. In value-function based reinforcement learning the agent maintains a value function $V : X \rightarrow \mathbb{R}$, which reflects the expected value of the discounted total rewards collected by starting from state x and following policy π:

$$V^{\pi}(x) := E\Big(\sum_{t=0}^{\infty} \gamma^t r_t \;\Big|\; x = x_0\Big) .$$

Let the optimal value function be

$$V^*(x) := \max_{\pi} V^{\pi}(x)$$

for each $x \in X$. If V^* is known, it is easy to find an optimal policy π^*, for which $V^{\pi^*} = V^*$. Value functions satisfy the famous Bellman equations:

$$V^{\pi}(x) = \sum_{a} \sum_{x'} \pi(x,a) P(x,a,x') \Big(R(x,a) + \gamma V^{\pi}(x') \Big) \tag{2.1}$$

and

$$V^*(x) = \max_{a} \sum_{x'} P(x,a,x') \Big(R(x,a) + \gamma V^*(x') \Big) . \tag{2.2}$$

An optimal policy can be defined by acting greedily according to V^*, that is, by selecting actions that maximise $V^*(x')$, the value of the next state:

$$a^*(x) \in \arg\max_{a} \sum_{x'} P(x,a,x') \Big(R(x,a) + \gamma V^*(x') \Big) .$$

One may also define a function of state-action values, or *Q-values*, expressing the expected value of the discounted total rewards collected by starting from state x and executing action a and following policy π onwards:

$$Q^{\pi}(x,a) = \sum_{x'} P(x,a,x') \Big(R(x,a) + \gamma V^{\pi}(x') \Big) .$$

Action selection then becomes $a^*(x) = \arg\max_a Q(s,a)$. It is also true that the optimal policy satisfies the following equation:

$$Q^*(x,a) = \sum_{x'} P(x,a,x') \Big(R(x,a) + \gamma \arg\max_{a'} Q^*(x',a') \Big) . \tag{2.3}$$

There are many alternatives to this setting. For example, one may use a reward function $R(x, x')$ depending on the current state x and the next state x', but not on the action executed. Also, one may replace action a by desired next state x_d, provided that there is an underlying sub-policy (not necessarily optimised) for reaching state x' from state x in one step [18].

Most algorithms that solve MDPs build on the Bellman equations [1]. In the following, we shall concentrate on the value iteration and the temporal difference learning algorithms. In tasks when the transition model P and the reward function R are not known a priori, the agent also needs to learn them on its own.

2.2 Value Iteration and Temporal Difference Learning

Value iteration uses the Bellman equations (2.2) as an iterative assignment. Starting from an arbitrary value function V_0 (represented as a table) it performs the update

$$V_{t+1}(x) := \max_a \sum_{x' \in X} P(x, a, x') \Big(R(x, a) + \gamma V_t(x') \Big)$$

for all $x \in X$. As it is well known (see e.g. [1]), the above update converges to a unique solution, and the solution satisfies the Bellman equations (2.2).

As this method updates *all* states at the same time, it is considered a *synchronous* algorithm. One *sampled, incremental* version of the algorithm can be obtained by updating only the values (by a small amount) of the states actually observed by the agent during its interaction with the environment, according to the following formula:

$$\begin{aligned} V_{t+1}(x) &:= (1 - \alpha) V_t(x) + \alpha \Big(R(x, a) + \gamma V_t(x') \Big) \\ &= V_t(x) + \alpha \Big(R(x, a) + \gamma V_t(x') - V_t(x) \Big) \\ &= V_t(x) + \alpha \delta_t \ , \end{aligned}$$

where α is an update rate and δ_t is the difference between the currently approximated value of the state x and its approximation based on the next state and the current reward, hence the name temporal difference. This is the so called temporal difference (TD) learning method.

The formula presented here is called the TD(0) method, as it only takes the immediate next state into account when updating the value of a state. It has been proven that this sampled version of the value function update is convergent. The method can be extended to longer time spans by means of the so called eligibility traces (TD(λ)) [16]. Note however, that TD learning is a policy evaluation method, thus it converges to the value function of a *fixed* policy, while value iteration converges the *optimal* value function. TD learning can be combined with optimistic policy iteration to guarantee convergence to the optimal value function (see [16]).

If the model parameters P and R are not known, they must also be learned from interaction with the environment; these functions are naturally sampled as

the agent interacts with the world. Estimation of the transition probabilities P can be performed by frequency counting, whereas the reward function R can be estimated by averaging if these quantities are represented as tables; i.e. separately for each state. Also, it is straightforward to rewrite the above update rules for Q values.

Unfortunately, despite their attractive convergence property, tabular methods are not applicable for real world tasks, where the state space is very large. The problem is that memory requirement grows exponentially with the number of the Cartesian variables and thus learning is slow. There is hope though: many of the states are similar from the point of view of the long term cumulated reward and one might try to generalise for similar states.

To this end, RL methods have been extended from tabular methods to function approximation to represent the value function (for an excellent introduction, see, e.g., [16]). For example, the value function can be expressed as linear combination of basis functions that characterise the states, or by more sophisticated techniques, including multi-layer neural networks. Unfortunately, convergence results are rare for function approximators: convergence can be proven for linear function approximators under certain conditions. Also, the quality of the function approximation heavily depends on the basis functions also called 'features'.

One thus prefers features that enable linear function approximation for the value functions *and* take advantage of the factorised nature of the state space. The value of a state variable in the successive time step is conditioned on the previous values of possibly a *few* other variables and the action taken. The reward function usually depends on a couple of variable combinations, i.e. the actual reward can be given by the actual value of a *few* variables. Knowledge about these dependencies enables us to construct relevant features for value function approximation, as it will be described in the next section.

2.3 Factored Markov Decision Processes

Consider a sequential decision process with m variables. In general, we need an exponentially large state space to model it as an MDP, thus the number of states is *exponential* in the size of the description of the task. Factored Markov decision processes offer a more compact task representation.

In the fMDP case one assumes that X is the Cartesian product of m smaller state spaces, corresponding to individual discrete variables having arbitrary (but finite) number of possible values:

$$X = X_1 \times X_2 \times \ldots \times X_m \ .$$

Continuous valued variables may also be fit into this frame by discretising them, and substituting with a discrete variable having as many possible values as the number of (disjoint) intervals used in the discretisation. The actual value of the new discrete variable become the index of the interval that the continuous value falls into.

Furthermore, let us call a function f a *local scope function* on X if f only depends on a (small) subset of the variables $\{X_i\}$. For the ease of notation, let us denote a set of variables by $X[I]$ and their corresponding instantiation by $x[I]$, where $I \subset \{1, \ldots, m\}$ is an index set.

Now, let us assume that we have an MDP on state space X. Then, the transition model of the factored MDP can be defined in a more compact manner than that of an MDP with states having no internal structure. The transition probability from one state to another can be obtained as the product of several simpler factors, by providing the transition probabilities for each variable X_i separately, depending on the previous values of itself and the other variables. In most cases, however, the next value of a variable does not depend on all of the variables; only on a few. Suppose that for each variable x_i there exist sets of indices Γ_i such that the value of x_i in the next time step depends only on the values of the variables $x[\Gamma_i]$ and the action a taken. Then we can write the transition probabilities in a factored form:

$$P(x, a, x') \equiv P(x' \mid x, a) = \prod_{i=1}^{m} P_i(x'_i \mid x[\Gamma_i], a)$$

for each $x, x' \in X$, $a \in A$, where each factor is a local-scope function

$$P_i : X[\Gamma_i] \times A \times X_i \to [0, 1] \qquad \text{for all } i \in \{1, \ldots, m\}. \tag{2.4}$$

By contrast, in the non-factored form, the probabilities of the components of x' can not be computed independently from subsets of all variables. Assuming that the number of variables of the local scope functions is small, then these functions can be stored in small tables. The size of these tables is a sharp (exponential) function of the number of variables in the Γ_i sets. These tables are essentially *conditional probability tables* of a dynamic Bayesian network (see e.g., [3]) of m variables.

The reward model of the factored MDP also assumes a more compact form provided that the reward function depends only on (the combination) of a few variables in the state space. Formally, the reward function is the sum of local-scope functions:

$$R(x, a) = \sum_{j=1}^{k} R_j(x[I_j], a),$$

with arbitrary (but preferably small) index sets I_j, and local-scope functions

$$R_j : X[I_j] \times A \to \mathbb{R} \qquad \text{for all } j \in \{1, \ldots, k\}.$$

The functions R_i might also be represented as small tables. If the maximum size of the appearing local scopes is bounded by some constant and independent of the number of variables m, then the description length of the factored MDP is polynomial disregard of the number of variables m.

To sum up, a factored Markov decision process is characterised by the parameters

$$\left(\{X_i\}_1^m\ ,\ A,\ \{\Gamma_i : P_i\}_1^m\ ,\ \{I_j : R_j\}_1^k\ ,\ \gamma\right).$$

The optimal value function can be represented by a table of size $\prod_{i=1}^{m} |X_i|$, one table entry for each state. To represent it more efficiently, we may rewrite it as the sum of local-scope functions with small domains. Unfortunately, in the general case, no exact factored form exists [8], however, we can still approximate the function by means of local scope functions:

$$\hat{V}(x) = \sum_{j=1}^{n} V_j(x[J_j]) \ . \tag{2.5}$$

The local scope functions V_j can be represented by tables of size $\prod_{i \in J_j} |X_i|$, which are small provided that the sets J_j are small, i.e., they involve only a few number of variables. The question is, how can we provide index sets J_j that are relevant for the approximation of the value function. If the local scopes Γ_i and I_j for the transition model and the reward model are known (which might be easy to define manually having sufficient knowledge about the task and the variables involved), we may use the following reasoning to deduce scopes for the value function: the value function is the long-term extended version of the reward function (whose index sets I_j are known). If we want to come up with an index set J_j of a local scope value function V_j which reflects long term values one step before reaching rewarding states, we need to examine which variables influence the variables in the set I_j. We can go on with this recursively to find ancestors of the variables in the set I_j, and iteratively determine the sets of variables that predict values on the long term. This process is called backprojection through the transition model [8].

We may rearrange the terms of the value function and redefine the linear approximation as follows. The table entries represent weights corresponding to *binary features*. Consider a local scope index set J_j, which means that the local value function V_j depends on the variables $\{X_i : i \in J_j\}$. Let $N_j = \prod_{i \in J_j} |X_i|$. We introduce binary features of the form $F_l(x) = \delta(\bigwedge_{i \in J_j} x_i = v_{l_i})$ for each possible value combination $\{(v_{l_1}, \ldots, v_{l_{|J_j|}}) \ , \ l = 1, \ldots, N_j\}$ of the variables $\{X_i : i \in J_j\}$. Function δ is the indicator function for condition c; $\delta(c) = 1$ if condition c is true, and 0 otherwise. That is, each table defines as many binary features as the size of the table. Then the value function approximation can be rewritten as:

$$\hat{V}(x) = \sum_{l=1}^{N} w_l F_l(x), \tag{2.6}$$

where $N = \sum_{j=1}^{n} N_j$, by reindexing the features F_l to run from 1 to N.

This form enables us to employ reinforcement learning techniques developed for linear function approximators. In this paper, we use (2.6) and apply temporal difference learning to factored Markov decision processes. The update of the parameters w is based on gradient descent utilising the temporal difference δ_t:

$$w_{t+1} = w_t + \alpha \delta_t \nabla_w V(x_t) = w_t + \alpha \delta_t F(x_t) \ , \tag{2.7}$$

where $F(x_t)$ is the vector of binary feature values for state x_t, α is the update rate. Eligibility traces and TD(λ) learning techniques are also applicable to linear function approximation (see [16] for an introduction).

It has been shown that the synchronous version of factored value iteration is convergent [17]. Moreover, when sampling is applied, the algorithm requires only a number of samples polynomial in the number of variables m (which can be much smaller than the number of states) to approximate the value function well with high probability. We expect – based on the close relationship between factored value iteration and factored temporal difference learning – that tabular temporal difference learning is also convergent, although we can not prove at the moment that convergence results for factored value iteration could be transferred to factored temporal difference learning.

When the factored model parameters, i.e. local scope functions P_i and R_i are unknown, they can be approximated from experience. The conditional probability tables corresponding to the local scope functions P_i can be updated separately by frequency counting for all variables and actions when observing state-to-state transitions. The factored reward function R can also be thought of as a linear function approximator, for example $R(x,a) = \sum_l u_l G_l(x,a)$ or $R(x,x') = \sum_l u_l G_l(x,x')$ based on some binary features G_l and parameters u_l (similarly to the value function), and can be updated using standard gradient descent techniques.

3 Related work

The exact solution of factored MDPs is infeasible. The idea of representing a large MDP using a factored model was first proposed by Koller and Parr [9]. More recently, the framework (and some of the algorithms) was extended to fMDPs with hybrid continuous-discrete variables [10] and factored partially observable MDPs [13]. Furthermore, the framework has also been applied to structured MDPs with alternative representations, e.g., relational MDPs [7] and first-order MDPs [14].

There are two major branches of algorithms for solving fMDPs: the first one approximates the value functions as decision trees, the other one makes use of linear programming.

Decision trees (or equivalently, decision lists) provide a way to represent the agent's policy compactly. Algorithms to evaluate and improve such policies, according to the policy iteration scheme have been worked out in the literature (see [9] and [2, 3]). Unfortunately, the size of the policies may grow exponentially even with a decision tree representation [3, 11].

The exact Bellman equations (2.2) can be transformed to an equivalent linear program with N variables $\{V(x) : x \in X\}$ and $N \cdot |A|$ constraints. In the approximate linear programming approach, we approximate the value function as a linear combination of basis functions (see, (2.5)), resulting in an approximate LP (ALP) with n variables $\{w_j : 1 \leq j \leq n\}$ and $N \cdot |A|$ constraints. Both the objective function and the constraints can be written in compact forms, exploiting the local-scope property of the appearing functions.

Markov decision processes were first formulated as LP tasks by Schweitzer [15]. The approximate LP form is a work of Farias [4]. Guestrin [8] shows that the maximum of local-scope functions can be computed by rephrasing the task as a non-

serial dynamic programming task and eliminating variables one by one. Therefore, the ALP can be transformed to an equivalent, more compact linear program. The gain may be exponential, but this is not necessary in all cases. Furthermore, the cost of the transformation may scale exponentially [5].

The approximate LP-based solution algorithm was worked out by Guestrin [8]. Primal-dual approximation technique to the linear program is applied by Dolgov [6], and improved results on several problems are reported.

The approximate policy iteration algorithm [9, 8] also uses an approximate LP reformulation, but it is based on the policy-evaluation Bellman equation (2.1). Policy-evaluation equations are, however, linear and do not contain the maximum operator, so there is no need for the second, costly transformation step. On the other hand, the algorithm needs an explicit decision tree representation of the policy. Liberatore [11] has shown that the size of the decision tree representation can grow exponentially.

4 Experiments

4.1 The scenario

The experiments reported in this paper were performed in a grid-world environment. This environment is part of an EC FET project, the 'New Ties project', which is a novel platform for multi-agent simulations. In the present simulations we experimented with single agents in order to evaluate our learning mechanisms, but the factored technique enables us to consider multi-agent scenarios in the future: agents can compute optimal behaviour by approximating other agents as additional factors.

The rectangular grid world contained two groups of food items at the far ends of the world. The task of the agent was to learn to consume food appropriately to survive: keep its energy level between two thresholds, that is, avoid being hungry, but also avoid being too much full; it received punishment for having the energy in the wrong ranges. Table 1 summarises the rewards of the agent depending on its energy level.

energy level	$\Delta_E < 0$	$\Delta_E = 0$	$\Delta_E > 0$
below lower threshold	-1	-1	Δ_E
in appropriate range	Δ_E	0	0
above upper threshold	$-\Delta_E$	-1	-1

Table 1: **Numerical values of the rewards.** Δ_E: change in the agent's energy. Additional component of the reward: cost for the agent's distance from home changed linearly in the range [0, 0.1].

The agent had a so called metabolism, so that it was better for the agent to consume both kind of food items, that is, if the agent consumed only one kind of

food, then its energy did not increase after a while. Also, we augmented the task with punishments for being far away from home, where home of the agent was its starting position in the grid world.

The agent was only able to observe the world partially, i.e. it had a cone of sight in front of it with a limited range. The agent moved on an 8-neighbourhood grid; it was able to turn left or right 45 degrees, and move forward. It had a cone of sight of 90 degrees in front of itself. It had a 'bag' of limited capacity, into which it was able to collect food items, and later consume the food from the bag. The primitive observations of the agent were food items in its cone of sight, its own level of energy and the number of food items in its bag of each type. The primitive actions were 'turning left/right', 'moving forward', 'picking up food to the bag', and 'eating food from the bag'.

4.2 Agent architecture

Since reinforcement learning in a heavily partially observable environment is very difficult in general and because the Markovian assumption on the state description is not met, we augmented the agent with high level variables and actions in order to transform the task and improve its Markovian properties. We note that there are formal approaches to tackle the problem of partial observability that aim to transform the series of observations automatically into a Markovian state description via belief states (see, e.g., [12] and the references therein), we did not choose to utilise the framework in the present study for the following reasons. First, we aimed to separate the factored MDP approach in a demanding scenario from the demands of partial observability. Second, the generation of high level features (state variables) from low level observations is a great challenge for artificial intelligence and is far from being solved in general and we wanted to gain insight into this problem through the scenarios. We do not list our negative experiences here, although they might be as important as the solution that we describe below.

The predefined high level variables were calculated from the history of observations and formed the variables of the state space of the factored MDP. The history of observations were stored using so called long term *memory maps*, for example one containing entries about where the agent has seen food items of a certain type in the past. Also, high level action macros were manually programmed as a series of primitive actions to facilitate navigation at a higher level of abstraction.

Figure 1 shows our agent architecture that makes use of high level variables and actions, and the factored architecture for value function approximation.

A sketch of the functioning of the agent architecture is shown in Algorithm 1. In the core of the algorithm is temporal difference learning with function approximation, for which memory maps serve as a preprocessing step to generate the current state of the agent from the history of observations. The agent also performs state transition and reward model learning.

Table 2 enumerates the high level variables and action macros that we used. In most cases the macros are related to variables; they can be chosen by the agent to alter the values of the variables, thus they are shown side by side in the table.

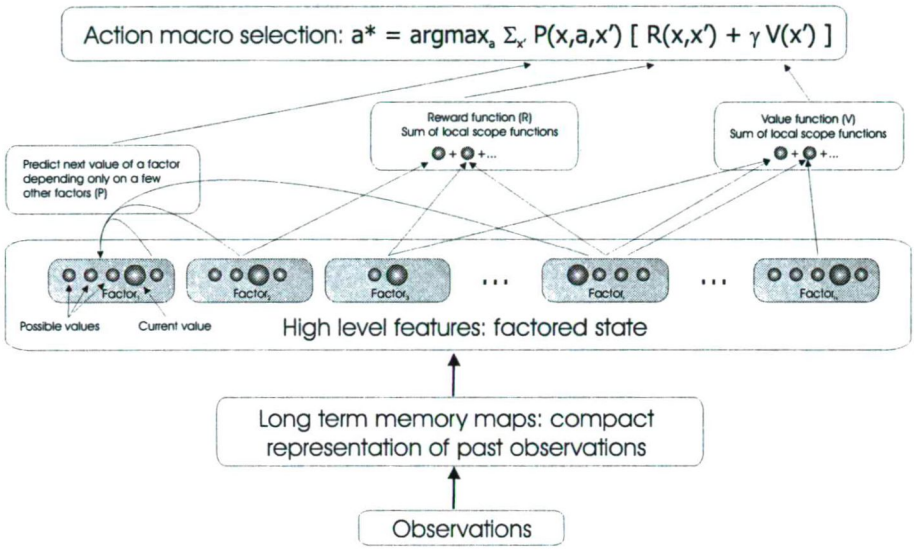


Figure 1: **Agent architecture.** The history of observations is summarised by long term memory maps. Based on these past and present observations, high level variables are generated, which form the variables of the factored MDP. The transition model (P) is learned as the product of local scope functions, and the reward (R) and value functions (V) are learned as the sum of local scope functions. Utilising these functions, action macro selection is accomplished in a greedy manner.

To make the description complete, we also need to provide the scopes of the local scope functions used. For the transition probabilities, this means providing the variables each state variable depends on, considering its next value when executing an action. For most variables, its next value depended only on its own previous value and the action taken, except for the energy level, which depended on itself, and the food history features as well. The reward function had factors depending on the energy level and the distance from home. The value function, which expresses long term cumulated rewards, had factors depending on the energy level, the number of food items in the bag, food consumption history, and the distance from home.

4.3 Experimental results

We compared three kind of reinforcement learning techniques to test the benefits of factorisation:

1. Q-table based learning (SARSA(λ)), no factorisation, only state-action values are learned, which implicitly incorporate transition probabilities.

Algorithm 1 : Agent life cycle. The agent applies temporal difference learning with linear function approximation extended with (factorised) model learning and memory map based preprocessing to generate states from observations

inputs:
state variables $\{X_i\}_1^m$, actions A,
local function scopes (index sets) $\{\Gamma_i\}_1^m$, $\{I_i\}_1^k$ and $\{J_i\}_1^n$ for transition probabilities, reward function and value function

1: **for** each time step t **do**
2: collect primitive observations, update long term memory maps
3: observe reward r_t for previous action or state transition
4: generate current state x_t (high level variables) using memory maps
5: update value approximation according to Eq. 2.7 or using TD(λ):
$w_{t+1} = w_t + \alpha\, F(x_t)[r_t + \gamma V(x_t) - V(x_{t-1})]$
6: update transition model parameters based on the observed state transition:
increase frequency counts for variable values in $x_{t-1} \rightarrow x_t$ upon a_{t-1},
recalculate probability values from frequency counts
7: update reward function approximation:
$u_{t+1} = u_t + \alpha\, G(x_{t-1}, x_t)[r - R(x_{t-1}, x_t)]$
8: choose next action:
$a_t = \arg\max_a \sum_{x'} P(x_t, a, x')[R(x_t, x') + \gamma V(x')]$
9: $t \leftarrow t + 1$
10: **end for**

2. V-table based learning (TD(λ)) along with the estimation of transition probabilities (P) and reward function (R) utilising tables. This step of factorisation separates state values form state transitions, i.e. from the effect of actions.

3. Factored learning (factored TD(λ)), where the V, R and P functions are factored utilising local scope functions.

Surprisingly, the state space described above already proved to be too large for the table based methods to make progress in learning in a reasonable amount of time. To make comparison possible, we had to reduce the state space to a minimal size; we reduced the number of discretisation intervals for some variables, and dismissed the feature 'distance from home' and the macro 'go back home'.

To show the learning process of the various methods, we calculated certain *learning curves* or *performance curves*: at each time step when the agent made a decision, we examined its energy level, and derived a series containing 1s and 0s, 1 meaning the energy level was between the two thresholds, 0 meaning it was not. By moving window averaging this series, this learning curve should tend to 1, provided that the agent learns to keep its energy between the two thresholds in a stable manner. We conducted the following experiments:

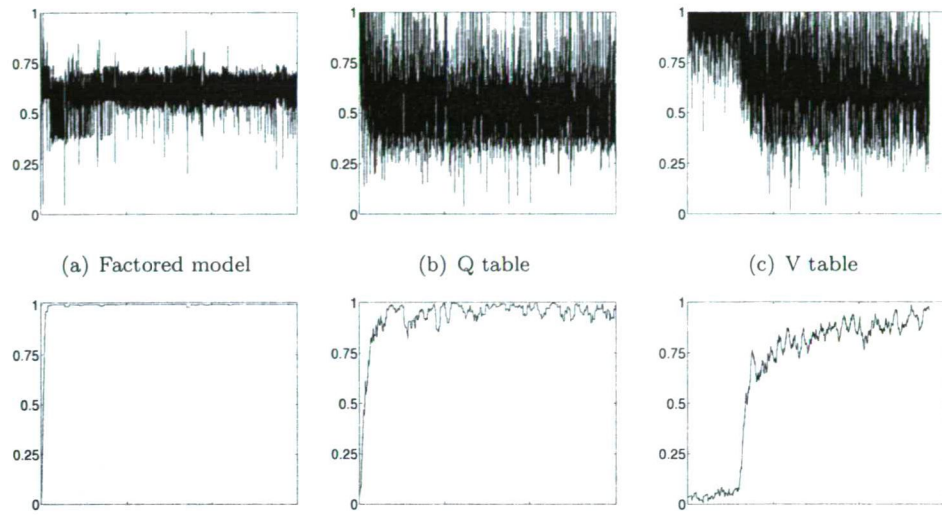
- We compared the energy curves of the agents during and after learning to examine how stable the behaviour of the agent was utilising the various methods.

High level variables	*Intervals or values*	*Notes*	*Action macro*
energy level	5	lowest and highest values are meant to be avoided	eat food, for each food type
number of food items in the bag	0-3	for each food type	collect food, for each food type
consumption history of food items	5	what fraction of the food consumed in the past few steps was of type t, for each food type	wait for a few time steps
distance to the nearest food item	5	for each type of food	explore; move in a random direction and amount
distance from home	5		go back home

Table 2: **High level variables and action macros used.** With these variables, (i) size of the state space is $5 \times 4^2 \times 5^2 \times 5^2 \times 5 = 250,000$, (ii) size of the state-action space is $5 \times 4^2 \times 5^2 \times 5^2 \times 5 \times 7 = 1,750,000$.

- We compared the learning curves of the various learning methods, to see how smooth the learning processes were, depending on the learning method.
- We tested how the various methods scale with the state space size.
- We tested the factored learning method in a slightly more complex setting, when we enabled the 'distance from home' feature and the 'go back home' macro. Also, in this setting the agent got penalised for being far from home, thus it had to optimise its behaviour according to two opposing objectives

In Figure 2 in the upper row, we show how the energy of the agent varies between the minimum and maximum values during a typical run for the three kinds of learning methods. The lower row shows the corresponding learning curves with a slight temporal smoothing. It can be seen that the factored model outperforms the table based methods both in learning speed and in its stability. The learning curve corresponding to the factored model goes up quickly to 1 right at the very beginning and stays there. The curve is smooth, demonstrating that the energy of the agent is kept between the two thresholds (which are 0.2 and 0.8). On the other hand, curves corresponding to table based methods rise towards 1 much more slowly and are much less stable, since the energy levels of the agents often exceed the thresholds. This comes from the fact that the factored model *generalises* very well, but table based models have no means to generalise, and need to learn the right actions for every possible combination of state variables.



(a) Factored model (b) Q table (c) V table

(d) Factored model (e) Q table (f) V table

Figure 2: **Typical energy levels and learning curves for the three learning methods.** The energy is the most stable for the factored model, for which learning is fastest and smoothest due to generalisation.

We also tested how the various methods scale with the state space size. To show the benefits of the factored approach, we increased the state space from minimal to a point until all methods could have been tested. In Figure 3, the curves are averaged over 10 runs and smoothed so that the learning trends could be seen. It can be seen that the learning time of the factored model is virtually not effected as the state space is increased, however, for table based methods, learning time increases greatly.

To see how the factored method behaves in a slightly more complex setting, we enabled the 'distance from home' feature and the 'go home' macro, and the agent also got punished based on its distance from home, to encourage it to stay near home, if possible. Note, that in this setting the agent had to optimise multiple criteria acting in opposite directions: to survive, it needs to get far from home, in order to get food, while at the same time it should spend as little time far from home as possible.

We examined the distribution of the agent's distance from home and concluded that it successfully learns to spend its time near home whereas it spent equal time at the two food areas when the feature was not enabled. In Figure 4(a) it can be seen, that if the agent is not punished for being far from home, it spends much time at the two ends of the world, which correspond to being close to home (one end of the world with one of the food sources) and being far from home (the other end

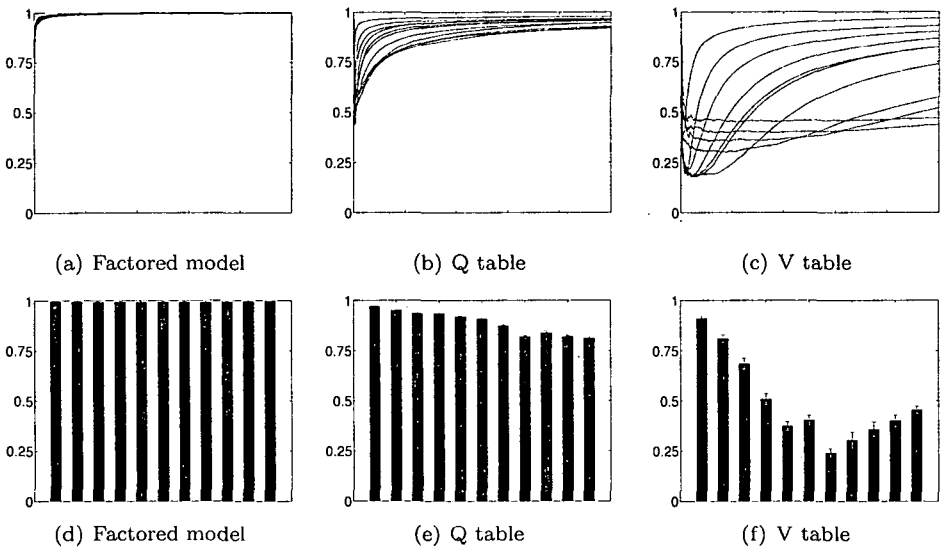
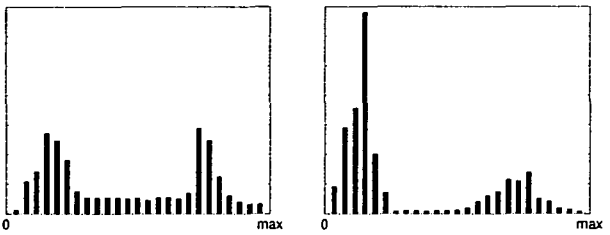


Figure 3: **The effect of varying state space size.** Bar plots: averaged performances and corresponding standard deviations of the methods after 20,000 steps for state space sizes 1,620; 3,645; 6,480; 11,520; 20,480; 32,000; 50,000; 112,500; 381,024; 1,075,648; 2,654,208 corresponding to the bars from left to right. Upper figures: corresponding learning curves up to 100,000 steps. The factored model is barely effected by the increase in the state space size. Learning time of the table based methods increases steadily, especially for V tables.

of the world with the other food source), and it spends medium amount of time in the area between the two ends. On the other hand, if it gets punished for being far from home, it spends much time near home, and it spends much less time at the other end of the world (b). Although the agent must occasionally visit the other end of the world in order to obtain the other kind of food, it can also be seen that the time spent in the middle of the world also gets lower, which suggests that the agent learned to quickly rush to the other end, get some food, and return home.

5 Discussion

We have investigated a learning model based on factored Markov decision process in a task which is real world-like in two ways. First, our agent lives in grid world in which it observes only a small neighbourhood of its environment. This *partial observability* usually entails the fact that the decision process related to the observations is not Markovian, i.e. past observations are also required to make the appropriate decisions. Second, the space of observations is overtly large. So in



(a) Locations w/o 'home' (b) Locations with 'home'

Figure 4: **Locations of the agent, shown by distance from home.** The agent should go to the far end to optimise for the metabolism. It learns to spend less time at the far end and by 'travelling' if it is penalised for being further away.

a sense, there are too many observations which are still not informative enough. To enable decision making, the agent needs to form a Markovian description of its state. To do so, we have utilised hand coded high level variables for spatial and temporal integration of observations based on long term memory maps. These variables build up the state of the agent.

However, such a state space is still too large for tabular RL methods even for the simple task described in this paper. This points to the need for methods of other type that can take advantage of the structure of the state space. The factored model builds exactly on the characteristics that the state space is generated as the Cartesian product of state variables. We have compared the factored method to traditional table based RL methods. In real-world tasks the agent usually needs to learn the model of the task, i.e. the state transition probabilities and the reward function as well. Q-table methods, on the other hand, naturally incorporate model learning. In our studies, we investigated how the separation of those learning subtasks effects learning speed. Thus, we compared (i) Q-table based learning to (ii) V-table based learning augmented with model learning, and to (iii) factored value-learning augmented with factored model learning.

The experiments demonstrated that the separation of model learning from value learning in the tabular case corrupts performance, i.e. the V-table based methods augmented with model learning were always worse than the Q-table based methods that incorporate model learning into (state-action) value learning. This is probably due to the fact that the Q-table based method needs to learn a much smaller number of parameters, because it does not rely on transition probabilities. Transition probability tables scale quadratically with the size of the state space, and these parameters are elegantly cumulated into a much fewer number of Q values. However, when the model is factored, the number of parameters describing the transition probabilities and the state values becomes much less, and it becomes beneficial to separate the model from the values; learning speeds up because of generalisation. Our experiments show that the factored model learns very quickly and becomes

stable very soon, since it is able to generalise knowledge learned in one state to an other.

6 Conclusion

We have experimented with a factored version of temporal difference based reinforcement learning. The partially observable nature of the task was diminished by hand crafted features, which generated the factored state space. We have shown that factored learning is faster and more stable as compared to tabular methods. Furthermore, the factored method is also applicable to large state spaces since it does not suffer from combinatorial explosion. The capability that transition probabilities can be learned for the factored case opens the way for planning in complex situations, such as the environment of the New Ties project, including the development and the execution of joint plans about desired future states. This makes this method promising for realistic applications.

References

[1] Bertsekas, D. and Tsitsiklis, J. *Neuro-dynamic programming.* Massachusetts Institute of Technology, 1996.

[2] Boutilier, C., Dearden, R., and Goldszmidt, M. Exploiting structure in policy construction. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, pages 1104–1111, 1995.

[3] Boutilier, C., Dearden, R., and Goldszmidt, M. Stochastic dynamic programming with factored representations. *Artificial Intelligence*, 121(1–2):49–107, 2000.

[4] de Farias., D. P. and van Roy, B. Approximate dynamic programming via linear programming. In *Advances in Neural Information Processing Systems 14*, pages 689–695, 2001.

[5] Dechter, R. Bucket elimination: A unifying framework for reasoning. *Artificial Intelligence*, 113(1–2):41–85, 1999.

[6] Dolgov, D. A. and Durfee, E. H. Symmetric primal-dual approximate linear programming for factored MDPs. In *Proceedings of the 9th International Symposium on Artificial Intelligence and Mathematics*, 2006.

[7] Guestrin, C., Koller, D., Gearhart, C., and Kanodia, N. Generalizing plans to new environments in relational MDPs. In *Eighteenth International Joint Conference on Artificial Intelligence*, 2003.

[8] Guestrin, C., Koller, D., Parr, R., and Venkataraman, S. Efficient solution algorithms for factored MDPs. *Journal of Artificial Intelligence Research*, 19:399–468, 2002.

[9] Koller, D. and Parr, R. Policy iteration for factored mdps. In *Proceedings of the 16th Conference on Uncertainty in Artificial Intelligence*, pages 326–334, 2000.

[10] Kveton, B., Hauskrecht, M., and Guestrin, C. Solving factored MDPs with hybrid state and action variables. *Journal of Artificial Intelligence Research*, 27:153–201, 2006.

[11] Liberatore, P. The size of MDP factored policies. In *Proceedings of the 18th National Conference on Artificial intelligence*, pages 267–272, 2002.

[12] Pineau, J., Gordon, G., and Thrun, S. Anytime point-based approximations for large POMDPs. *Journal of Artificial Intelligence Research*, 27:335–380, 2006.

[13] Sallans, B. *Reinforcement Learning for Factored Markov Decision Processes.* PhD thesis, University of Toronto, 2002.

[14] Sanner, S. and Boutilier, C. Approximate linear programming for first-order MDPs. In *Proceedings of the 21th Annual Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 509–517, 2005.

[15] Schweitzer, P. J. and Seidmann, A. Generalized polynomial approximations in markovian decision processes. *Journal of Mathematical Analysis and Applications*, 110(6):568–582, 1985.

[16] Sutton, R. and Barto, A. G. *Reinforcement Learning: An Introduction.* MIT Press, Cambridge, MA, 1998.

[17] Szita, I. and Lőrincz, A. Factored value iteration converges. *Acta Cybernetica*, 18(4):615–635, 2008.

[18] Szita, I., Takács, B., and Lőrincz, A. Epsilon-MDPs: Learning in varying environments. *Journal of Machine Learning Research*, 3:145–174, 2003.

Acta Cybernetica 18 (2008) 669–680.

An Evolutionary Algorithm for Surface Modification

Evgeny Lomonosov* and Gábor Renner*

Abstract

Complex shapes bounded by free form surfaces are designed and modified by specifying the parameters of their mathematical descriptions. Although most graphics and design systems provide high level tools for free form shape modifications, they are not intuitive and suitable for a creative designer or a stylist. In the present paper a technique is proposed for the modification of the shape of a parametric B-spline surface through embedding a set of predefined characteristic curves. Shape modification is performed by an evolutionary algorithm. The evolutionary algorithm minimises the distance between the curve and the surface, while maintaining the global shape and smoothness of the original surface. Several experimental results are presented.

Keywords: shape modification, evolutionary algorithms, B-spline surfaces

1 Introduction

Complex shapes are described as free form surfaces and represented in parametric forms such as Bézier, B-spline or NURBS surfaces in computer graphics, computational geometry, and engineering design (CAD/CAM). They are usually created and modified by defining and modifying parameters of their mathematical representations, mainly the control points, eventually degrees, weights and knot vectors. Working with these requires mathematical knowledge and skills and is not suited to the engineering concepts and practice of shape design. Shape modification should be intuitive, easy to define, and resulting in the expected shape. Additionally, in engineering practice, acceptable compromise has to be found between different, sometimes contradictory requirements concerning shape and dimensions. Because of functionality, the resulting shape must meet tight dimensionality and shape constraints. It must also be aesthetically pleasing, free from unwanted bumps and wiggles. All these expectations can only be realised in a highly iterative process, which is costly and time consuming.

Methods for intuitively designing and modifying complex free form surfaces are described in several research papers. They try to control the shape by geometrical constraints and user parameters that are easy to manipulate. The so called

*MTA SZTAKI, Budapest, Hungary. E-mail: {elomonosov,renner}@sztaki.hu

Dynamic NURBS [9] introduces mass distribution, deformation energies and forces into the geometrical model in order to deform surfaces in a physically intuitive manner. In case of the so called variational design tool method, modification is governed by an underlying mechanical model; a bar network which is associated with the control polygon of the surface [3]. Due to the complexity of the deformation process, the allowable geometrical constraints are usually simple, for example, moving the surface to become tangent to a prescribed plane. Line constraints are more appropriate for surface modifications; they have been introduced into variational design by Pernot et al. [6]. Design tools of this kind can be integrated into a free form feature concept, providing high level tools for shape modifications [5]. Recently haptic tools were suggested for the direct manipulation of physically based B-spline surfaces [1].

A basic problem with the above approaches is that the effect of the modification is usually not precisely defined. They work well in the design phase, mostly in conceptual design. However, in many cases of shape modification, e.g. when redesigning an already existing object, well defined modifications are needed with precise shape and dimensionality constraints. Because of the free form characteristics of the surfaces, the modification must still be flexible, not constrained by the features of the mathematical description.

A typical example of the above type of shape modification is when a free form surface must be created which embeds a previously defined space curve. The designer usually starts with an initial shape, and the modified shape must be similar to the initial one as much as possible.

The algebraic complexity of free form curves lying on free form surfaces is high [8], and there is no real chance to embed analytically an arbitrary space curve into a smooth surface. However we can try to embed a good approximation of the curve, which also provides the possibility to satisfy additional design constraints, such as the definition of the surface region to be modified. Alternatively, the requirement of minimal distortion or of maximal smoothness of the final surface can be used.

These kinds of shape modification problems can be handled successfully by combining geometrical and numerical computations with evolutionary search methods. In this paper a method based on evolutionary algorithms is proposed to modify the shape of a complex surface with the constraint of embedding a space curve.

2 Evolutionary algorithms

Evolutionary algorithms are powerful and robust search procedures based on artificial adaptation. Evolutionary algorithms search for the solution of a problem simulating a process of evolution in nature. The problem is formulated as a search for the global maximum (or minimum, for some problems) of a given function which is called the fitness function. A fitness function depends on a set of parameters characterising the problem. It measures how fit is a potential solution, as specified by some combination of parameter values, for our purposes. The aim is to find the most suitable combination which corresponds to the maximal (or minimal) value

of the fitness function.

Conventional search techniques usually process one candidate solution a time. In contrast with these, evolutionary algorithms consider several potential solutions at once. A number of potential solutions form a population. An individual in a population represents one possible combination of parameter values. By analogy to natural genetics, the representation of a solution for an evolutionary algorithm is called a chromosome. A chromosome consists of genes which usually directly represent the parameters of the problem. Alternatively, an encoded representation may be used, for instance, a binary encoding proposed in [2]. It is often advantageous to incorporate problem specific knowledge into the problem representation for evolutionary algorithms.

Random numbers are normally used to generate an initial population. Another approach is to apply some heuristics to ensure that the individuals of the initial population cluster in those regions of the parameter space where the final solution is likely to be found. Subsequent generations are formed by selecting individuals with better fitness function values. The selected individuals are called parents. Individuals of the next generation (called offsprings) are obtained by applying genetic operators to parents. Similarly to the way it happens in nature, offsprings inherit properties of their parents and more fit individuals are more likely to pass their genes to the next generation.

A number of methods may be used to select parents, but it is essential that individuals with better fitness values have more chances to be reproduced in the next generation. Most commonly used genetic operators include crossover and mutation. The crossover operator exchanges genes between parents to form an offspring. The mutation operator changes some genes randomly. The evolutionary process is controlled by a number of parameters, the most important of which are the population size and the rates of mutation and crossover. As more fit individuals have more chances to pass their genes to the new generation, average fitness gradually improves. The pseudocode summarising the algorithm is shown below.

Algorithm 1 Evolutionary algorithm

```
1: Generate initial population
2: while Termination condition not satisfied do
3:    Evaluate fitness functions of individuals
4:    Select pairs of individuals to become parents
5:    Create offspring using crossover operator
6:    Apply mutation operator to offspring
7: end while
```

3 Surface shape modification

In this paper we consider the task of modifying an existing free-form surface to satisfy some constraints given as a number of free-form curves that should be em-

bedded into the resulting modified surface. The resulting surface should also retain similarity to the initial one and at the same time be smooth and visually pleasing. The surface is given by B-spline representation defined by the expression:

$$\mathbf{S}(u,v) = \sum_{i=0}^{m}\sum_{j=0}^{n} \mathbf{p}_{ij} N_{ik}(u) N_{jl}(v), \tag{1}$$

where $\mathbf{p}_{ij}$ are control points forming a control net and $N_{ik}(u)$, $N_{jl}(v)$ are B-spline basis functions of order k and l [4].

The curves to be embedded into the surface can be given in an arbitrary representation commonly used in computer aided geometric design. In our experiments a cubic B-spline curve was used. Because exact representation of curves on surfaces has high algebraic complexity [8], we do not aim at producing a surface which embeds the given curve algebraically. Instead, the curve is sampled and the sampled points are used to calculate the fitness function. Curve sampling makes it possible to define the distance between a curve and a surface. The fitness function to be minimised is then defined as the sum of squared distances from the sampled points on the curve to the nearest points on the surface:

$$f(S) = \sum_{i=0}^{N} d^2(\mathbf{c}_i, \mathbf{s}_i), \tag{2}$$

where N is the number of sampled points, $\mathbf{c}_i$ is a sampled point on the curve, and $\mathbf{s}_i$ is the nearest point on the surface. A rough initial estimate and Newton-Raphson iteration are used to find $\mathbf{s}_i$.

All the parameters defining a B-spline surface can be used to modify its shape. The degree can be changed in each parametric direction, knots and control points can be moved, inserted, or removed. If the algorithm was allowed to change all the parameters of the surface it would lead to a very large search space and slow algorithm execution. To simplify the search space and to make the algorithm faster, we only consider moving control points in this paper, leaving the degrees unchanged and the knot vectors inherited from the initial surface.

An important property of the B-spline surface representation is its locality in the sense that a control point has influence only on a well defined region of the surface. As our goal is to obtain a surface which is similar to the initial one as much as possible, it is reasonable to alter only the patches of the surface nearest to the curve. Therefore the search space is further reduced by considering only control points which have influence on patches of the surface located near the curve. For each of the sampled curve points we find the nearest point on the surface and the corresponding knot intervals in the parameter space of the surface. Next, using the relations between control points and local basis functions, we determine the control points which have influence on this region of the surface. Alternatively, the region to be modified can be defined in an interactive way.

In Figure 1 an example of a surface is shown (a car body element) with parametric lines and modifying curves. Patches of the surface that have to be modified

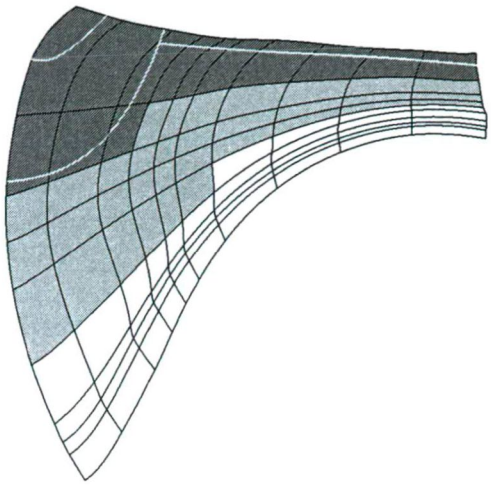


Figure 1: Modified patches

to embed the curve are shown in dark gray. Other patches that were affected by moving control points are shown in light gray and patches that were not modified are shown in white.

Some authors (e.g. Watabe [10]) suggest using multidimensional arrays as chromosomes when working with multidimensional data. However, our experiments showed that a one-dimensional array, where each gene corresponds to a displacement of one particular control point in the two-dimensional control net, performs better in our task. This can be attributed to the fact that in our case only part of the control net is varied by the algorithm, and the shape of this part is often not rectangular. Therefore using rectangular two-dimensional arrays as chromosomes would force the algorithm to process a lot of meaningless data, making it considerably slower.

As a consequence of the above experience and argumentation, a one-dimensional array of real-valued 3D vectors was used as the genetic representation (chromosome). Each vector corresponds to the displacement of one of the affected control points with respect to its initial position. Each individual surface is therefore defined by

$$\mathbf{S}(u,v) = \sum_{i=0}^{m}\sum_{j=0}^{n}(\mathbf{p}_{ij} + \Delta\mathbf{p}_k)N_{il}(u)N_{jm}(v), \tag{3}$$

where $\Delta\mathbf{p}_k$ are elements of the chromosomes (genes), $k = k(i,j)$. Mapping of genes onto the control net is illustrated in Figure 2.

New generations are formed using tournament selection of individuals. Two individuals are selected at random from the current population. Their fitness values are compared, and the individual with a better fitness value is used as a parent.

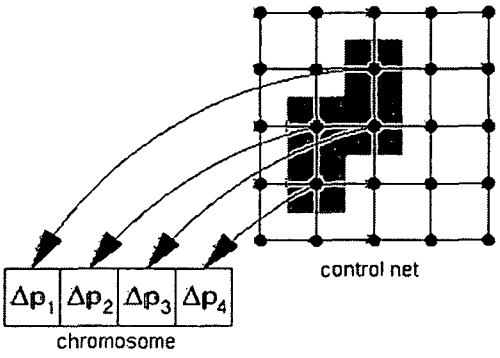


Figure 2: One-dimensional array of displacement vectors is mapped onto two-dimensional control net.

The second parent is selected in the same way. Crossover is performed as a random exchange of genes between the two parents. Any gene in the offspring has an equal probability of coming from any of the two parents.

Two mutation operators are used in the algorithm. The first mutation operator changes each component of the current displacement vector by a value not exceeding 10% of the allowed range. The other mutation operator replaces the current displacement vector in the chromosome with a randomly generated new one. In both cases each component of the new displacement vector is constrained to lie in the allowed range.

The allowed range for components of any displacement vectors is defined as a multiple of the maximum distance between a sampled point on the curve and the initial surface:

$$\Delta_{max} = C_\Delta \cdot max(d^2(\mathbf{c}_i, \mathbf{s}_i)), \tag{4}$$

where C_Δ is a coefficient. This effectively constrains control points of the resulting surface to lie inside a cube with the centre in the initial position of the point and with an edge size of $2\Delta_{max}$.

In order to make the best fitness non-increasing, elitism is used in the algorithm. A copy of the most fit individual in the current generation is preserved in the next generation with no crossover and mutation.

Besides embedding the curve, the resulting modified surface should retain similarity to the original one, being as close to it as possible. This is achieved without incorporating additional constraints in the algorithm or adding extra terms to the fitness function, by setting all displacement vectors in the initial population to zeros. Preliminary experiments showed that the algorithm that starts from randomly generated population, tends to move control points more than it is necessary, thus producing surface that is far from the initial.

The algorithm is terminated if the best fitness does not change for a number of generations or reaches a desired threshold value.

4 Surface smoothing

Many different methods can be applied to generate smooth B-spline curves or surfaces within an evolutionary process (see e.g. [7]). For instance, a smoothness criterion can be incorporated into the fitness function. The evolutionary algorithm, in this case, optimises the smoothness of the surface simultaneously with the above distance function. The resulting surface is a compromise between the desired shape and smoothness. In our case, incorporating a smoothness measure into the fitness function did not yield positive results. A possible explanation is that minimising the smoothness metric for the surface attempts to control smoothness globally, while we only need to preserve smoothness in the modified region and its surroundings. Control point modifications necessary for embedding the curve tend to spoil smoothness in these regions.

Another way to ensure surface smoothness is to perform additional smoothing as a separate post processing step, not taking smoothness into account when embedding curves. The drawback of this method is that the distance function usually deteriorates at the post processing step, and we can lose the desired shape of the surface. It may be more appropriate to control smoothness during the process of distance optimisation.

We developed a method that allows local control of surface smoothing. The proposed method tries to balance the change in the surface smoothness, which is introduced whenever the algorithm moves a control point, by means of moving other control points. If the original surface is smooth, embedding the curves which do not belong to the surface often makes the surface more bumpy. We are trying to compensate for this by moving neighbouring points whenever a control point is moved by the algorithm, to minimise changes in curvature as illustrated in Figure 3.

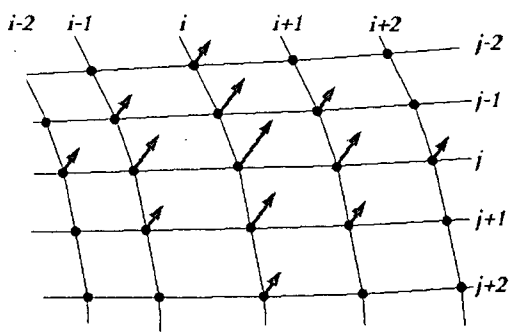


Figure 3: When a control point is moved, its neighbours also move.

If the algorithm moves point $\mathbf{p}_{ij}$ by $\Delta\mathbf{p}_k$, then neighbouring control points $\mathbf{p}_{i-1,j}$, $\mathbf{p}_{i+1,j}$, $\mathbf{p}_{i,j-1}$ and $\mathbf{p}_{i,j+1}$ are moved automatically by $2/3\ \Delta\mathbf{p}_k$. Furthermore, control points $\mathbf{p}_{i-2,j}$, $\mathbf{p}_{i+2,j}$, $\mathbf{p}_{i,j-2}$, $\mathbf{p}_{i,j+2}$, $\mathbf{p}_{i-1,j-1}$, $\mathbf{p}_{i-1,j+1}$, $\mathbf{p}_{i+1,j-1}$ and $\mathbf{p}_{i+1,j+1}$ are also moved, and their displacement is $1/3\ \Delta\mathbf{p}_k$. In this way we avoid

excessive relative control point movements. Therefore the algorithm produces a surface which is more visually pleasing and more similar to the initial one than those obtained by moving control points independently.

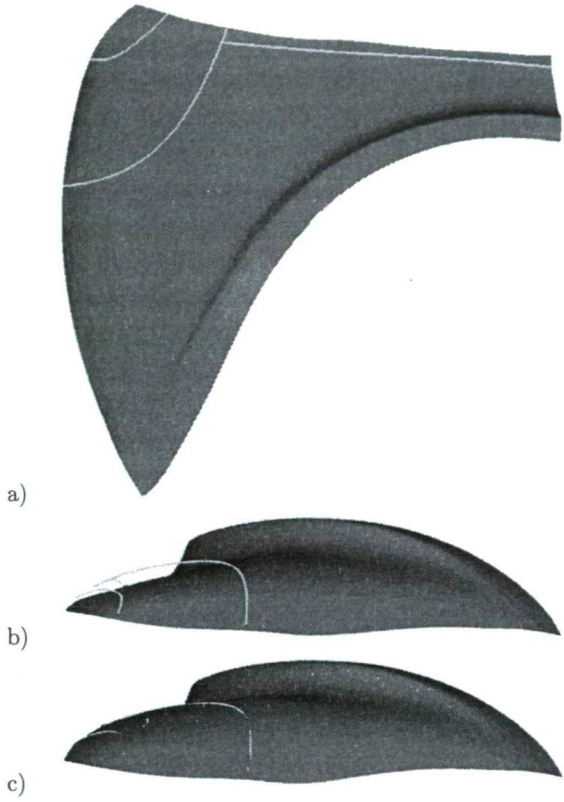


Figure 4: A car body part with modifying curves a) Front view b) Side view before the modification c) Side view after the modification

5 Results

The algorithm was applied to a number of different surfaces from various application areas, including technical and medical objects. Modifying curves of different shapes and spatial position were designed to guide the modification process. The aim was to obtain a surface that embeds the modifying curve and at the same time retains smoothness and shape similarity to the initial surface.

All the surfaces that we used were reverse-engineered from measured points using the GEOMAGIC Fashion 6 surface fitting package. The resulting surfaces

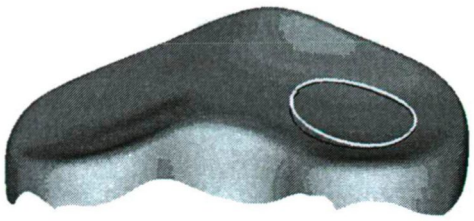


Figure 5: A medical replacement with a modifying curve (shown in white) and its projection onto the surface (shown in black)

are 3rd degree non-uniform polynomial B-spline surfaces. The modifying curves were designed using the Blender three-dimensional modelling system.

One of the most important application fields for our algorithm is automobile design. Therefore we used a real part of a car body, as the first surface for our experiments (Figure 4). We consider the situation when the designer alters the shape of an existing car body part to satisfy new functional and aesthetic constraints. For better visualisation of the relationship between the surface and the modifying curves we show both the front view and the side view. The result of surface shape modification is shown in Figure 4(c).

Another possible application field for surface modification algorithms is the reshaping of medical replacements (e.g. knee prostheses) in order to fit them to the shape and size of the patient. Figure 5 shows the tibia (lower limb) part of a knee prothesis with a modifying curve above the surface and its projection onto the surface. The third example is a sheet metal part surface which was chosen for its complexity and high curvatures (Figure 6).

For all the data sets, similar parameters were used in the evolutionary process, which were chosen after some preliminary experimentation. The crossover rate was set to 0.9. The mutation rate was 0.4 for the first mutation operator, and 0.05 for the second. The population size was set to 50 in all the cases.

For the car body example, both the original and the modified surfaces are shown, as the difference can be seen clearly. For the other two examples the resulting surface is very similar to the original one, and the difference cannot readily be seen. Therefore, the performance of the algorithm should be evaluated using numerical results.

The results of the experiments are shown in Table 1. Because of its stochastic nature, the evolutionary algorithm produces slightly different results each time we run it. Therefore, ten experiments were made with each surface. Table 1 shows the average values for ten runs of the algorithm. The following values are shown in the table:

1. the dataset name,
2. the number of control points of the surface,

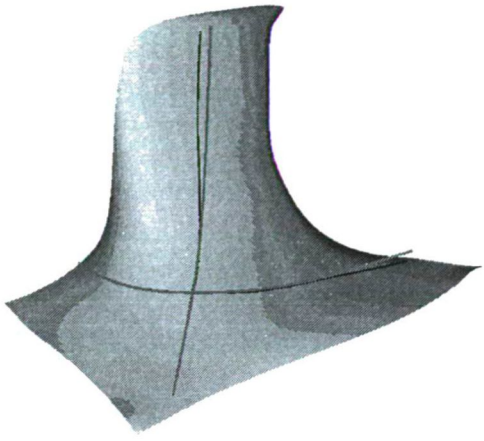


Figure 6: A metal sheet with a modifying curve (grey) and its projection (black)

3. the number of control points in the modified region,
4. the average number of generations before the fitness function reaches 0.1% of its starting value,
5. the average number of generations before the fitness function value stops changing,
6. the initial value of the fitness function (according to (2)),
7. the average final value of the fitness function,
8. the maximal final value of the fitness function.

Table 1: Numerical results of the experiments

1	Dataset name	Car body	Medical replacement	Metal sheet
2	Num. ctrl. points	156 (12 × 13)	238 (14 × 17)	72 (8 × 9)
3	Num. ctrl. points modified	58	56	59
4	Num. gen. before threshold	596	181	588
5	Num. gen. before stop	1874	1018	1944
6	Initial fitness value	6781.8	35.55	1128.8
7	Avg. final fitness value	2.6319	0.0003	0.3185
8	Max. final fitness value	4.2354	0.0006	0.4358

Two different termination conditions were employed in our experiments. The algorithm was terminated when the fitness function reached a pre-defined threshold, which was set to 0.1% of the initial fitness function value. The average number

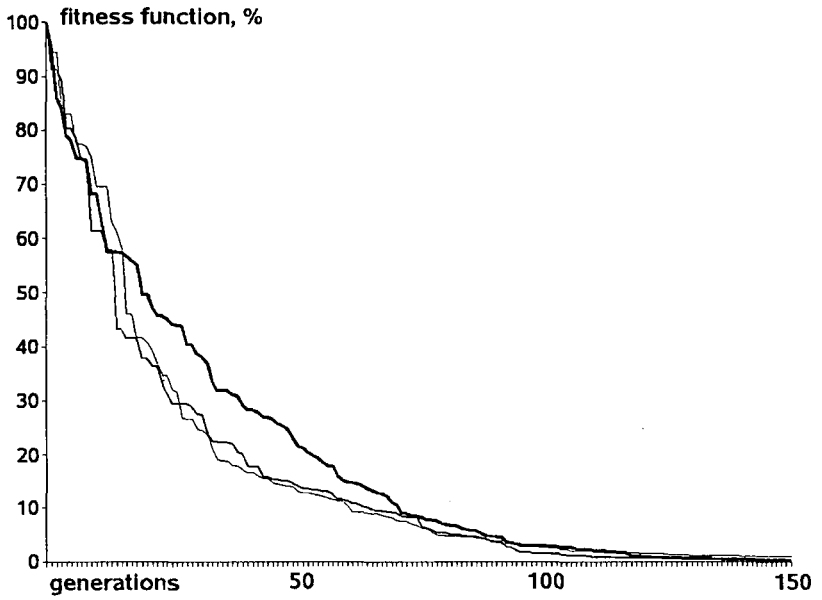


Figure 7: Fitness function change with generations

of generations needed to reach the threshold is given in the 4th row of Table 1. Then we continued the execution of the algorithm until the fitness function stopped changing its value. In practice, we terminated the algorithm as soon as the fitness function value did not change for 15 generations. The average number of generations before the fitness function value stopped changing is shown in the 5th row of the Table 1, the average final value of the fitness function in the 7th row, and the maximal final value of the fitness function in the 8th row.

Figure 7 shows how the the fitness function changes with the generations. Fitness is displayed here as a percentage of the initial value. Three typical cases were chosen, one for each example. It can be seen that an acceptable result was obtained in all the cases by the 150th generation. This result could later be improved only slightly.

It can be seen from Table 1 and Figure 7 that the algorithm reduced the fitness function value by three to five orders of magnitude. The resulting fitness function value is well below the required technical tolerance.

6 Conclusions

Free form shapes are modified in CAD/CAM systems by the parameters of their mathematical representations, mainly control points. This technique is tedious and not intuitive for a creative designer. In the paper a method is proposed to modify

the shape of a B-spline surface in order to embed a previously given characteristic curve into the surface. Control points of the surface are moved by an evolutionary algorithm. Genetic representation and control point modifications are constructed so as to minimise the distance between the curve and the surface, together with the changes in surface smoothness. Experimental results demonstrated the applicability of our method to several types of surfaces and curves. Examples from car body design and medical applications were given. Searching for other field of applications is in progress.

References

[1] Dachille, Frank, Qin, Hong, Kaufman, Arie, and El-Sana, Jihad. Haptic sculpting of dynamic surfaces, April 1999.

[2] Goldberg, David E. *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Wesley Publishing Company, Reading, Massachusetts, 1989.

[3] Guillet, S. and Léon, J.C. Parametrically deformed free-form surfaces as part of a variational model. *Computer Aided Design*, 30:621–630, 1998.

[4] Hoschek, Joseph and Lasser, Dieter. *Fundamentals of Computer Aided Geometric Design.* A K Peters, 1993.

[5] Pernot, J-P., Falcidieno, B., Giannini, F., Guillet, S., and Léon, J-C. Modelling free-form surfaces using a feature-based approach. In *SM '03: Proceedings of the eighth ACM symposium on Solid modeling and applications*, pages 270–273, New York, NY, USA, 2003. ACM.

[6] Pernot, Jean-Philipe, Guillet, Stephane, Léon, Jean-Claude, Giannini, Franca, Catalano, Chiara Eva, and Falcidieno, Bianca. A shape deformation tool to model character lines in the early design phases. In *SMI '02: Proceedings of the Shape Modeling International 2002 (SMI'02)*, page 165, Washington, DC, USA, 2002. IEEE Computer Society.

[7] Renner, Gábor and Ekárt, Anikó. Genetic algorithms in computer aided design. *Computer-Aided Design*, 35(8):709–726, 2003.

[8] Renner, Gábor and Weiss, Volker. Exact and approximate computation of B-spline curves on surfaces. *Computer-Aided Design*, 36(4):351–362, 2004.

[9] Terzopoulos, Demetri and Qin, Hong. Dynamic NURBS with geometric constraints for interactive sculpting. *ACM Transactions on Graphics*, 13(2):103–136, 1994.

[10] Watabe, Hirokazu and Okino, Norio. A study on genetic shape design. In *Proceedings of the 5th International Conference on Genetic Algorithms, Urbana-Champaign, IL, USA, June 1993*, pages 445–451, 1993.

Acta Cybernetica 18 (2008) 681–693.

Investigating the Behaviour of a Discrete Retrial System

Péter Kárász*

Abstract

Certain technological applications use queuing systems where the service time of entering entities cannot take any value, it can only be a multiple of a certain cycle-time. As examples of this, one can mention the landing of aeroplanes and the optical buffers of internet networks. Servicing an entering customer can be started immediately, or, if the server is busy, or there are waiting customers, the new customer joins a queue, moving along a closed path which can be completed within a fixed cycle-time of T units. Applications in digital technology induce the investigation of discrete systems. We give the mathematical description of systems serving two types of customers, where inter-arrival times follow a geometric distribution, and service times are distributed uniformly. A Markov-chain is defined and the generating functions of transition probabilities are calculated. The condition of ergodicity is established and the equilibrium distribution is given.

Keywords: discrete retrial systems, Lakatos-type queuing system

1 Introduction

The system investigated in the paper was originally based on a real problem connected with the landing of aeroplanes, but later many other applications emerged which are strongly related to information technology. We use the original problem to provide an initial description.

Consider an airport where aeroplanes come to land. The airport can serve only one plane at a time. Hence, if the runway is used or there are other planes waiting to land, the incoming plane has to wait. Unlike in classic queuing systems, special conditions prevail here, which result in significant differences from an ordinary system. We assume that a plane planning to land approaches the runway in the optimum position, and, if it is possible, it starts landing immediately. If the plane is forced to wait, then it starts a circular manoeuvre and can issue further requests to land only when reaching the original starting point of its trajectory. We assume

*Budapest Tech, John von Neumann Faculty of Informatics, Bécsi út 96/b, H–1034 Budapest, Hungary, E-mail: karasz.peter@nik.bmf.hu

that completing each full cycle takes equal time, T, thus the possible instants of starting to land can differ from the moment of arrival by integer multiples of T. Because of possible fuel shortage it is natural to use the first-come-first-served (FCFS) discipline.

We can model the above problem by investigating a retrial system, where the service of an incoming customer can be started upon arrival if the system is in free state, otherwise – if the server is busy or there are other entities waiting – the incoming customer joins a queue and its service is started at the nearest possible instant differing from the arrival by a multiple of the cycle-time T. The FCFS rule is obeyed. Different service time distributions lead to various problems, these were broadly investigated by Lakatos. In [9], service time distribution is exponential, whereas in [10] it is uniform. In the light of technical applications, it is important to consider discrete models. In this case the cycle-time is divided into n equal time-slices, which form the basis of the discrete distributions. A typical application in digital technology is the use of an optical buffer, which is a device capable of temporarily storing light (or rather, data in the form of light). As light cannot be frozen, a typical optical buffer is realised by a single loop, in which data circulate a variable number of times, and thus n can be the measure of the cycle-time in clock cycles. The model was investigated by Rogiest, Laevens et al. in [8, 14]. The involved optical buffers are implemented as a set of $N+1$ Fiber Delay Lines (FDLs), with lengths that are typically multiples of a basic value D, called the granularity. This results in a degenerate waiting room with waiting times 0, D, $2D$, ..., ND. The problem is investigated from the point of view of the customers, i. e. their waiting time. Lakatos and myself chose a different approach, the problem is described from the aspect of the server, i. e. the number of waiting customers, which is more significant for determining the number of necessary FDLs. The time elapsed between two arrivals was geometrically distributed and service times of customers were geometric and uniform in [11] and [12], respectively. The two different approaches to describe these systems coincide in the condition of ergodicity and the probability of free state; this was shown in [13]. A numerical investigation was carried out in [2].

It was Kovalenko's suggestion to generalise the problem for two different types of customers. Only one customer of the first type can be present in the system. Such a customer can be accepted for service only in the case of a free system, in all other cases its request is turned down. There is no such restriction for the customers of the second type; they are serviced immediately or join a queue, when the server is busy. This type of system was examined with different continuous distributions in [4, 5, 6]; simulation results were also included in [6]. In [7] this type of system using relative priorities was investigated with geometric inter-arrival and service time distributions. In the present paper the same system is considered with discrete uniform service time distributions. The endpoints of the interval of the uniform distributions are presumed to be multiples of the cycle-time T. This assumption does not restrict the generality of the theory, but without it formulae are much more complicated.

There are several aspects which can be treated just exactly in the same way as in

the case of continuous distributions. However, there are some differences between discrete and continuous systems. One such significant phenomenon is collision. With continuously distributed service times the probability of the appearance of two different types of customers at the very same instant is zero, but in discrete systems different types of customers do appear during the same time-slice with non-zero probability. There are several ways to deal with collisions, we suggest three methods for treating these, numbered I., II., and III.

The aim is to determine generating functions of transition probabilities, as well as to establish the condition of ergodicity.

2 Results

Consider a Lakatos-type queuing system serving two types of customers. The cycle-time T is divided into n equal time-slices. The probability of the appearance of a customer of type j during a certain time-slice is r_j, and there is no entry with probability $1 - r_j$, i. e. inter-arrival times are geometrically distributed with parameters r_j $(j = 1, 2)$. Service times are uniformly distributed in the interval $[\gamma_j, \delta_j]$, (where γ_j and δ_j are multiples of T), i. e. the probability that the service time of a customer of type j is in this interval is $q_j = \frac{T}{n(\delta_j - \gamma_j)}$.

For the description of the system we are going to use the *embedded Markov-chain* technique. Let us consider the number of customers in the system at the moment just before the service of a new customer begins. In other words, if t_k denotes the moment when the service of the k-th entity starts, we consider the sequence, whose states correspond to the number of customers at $t_k - 0$. For the sake of definiteness, at $t = 0$ let the system be free. To see that the process is Markovian we refer to the same argument as in [6], and the memoryless property of the geometric distribution.

For this chain we introduce the following transition probabilities:

a_{ji}: the probability of the appearance of i customers of the second type at the service of a type j customer $(j = 1, 2)$, if at the beginning there is only one customer in the system;

b_i: the probability of the appearance of i customers of the second type at the service of a second type customer, if at the beginning of the service there are at least two customers in the system;

c_i: the probability of the appearance of i customers of the second type in a free state.

As the process runs, the busy period can start with a customer of either type. During the service of this customer only second type customers are accepted for service, they join the queue. Requests of first type customers are refused. This explains the need for introducing c_i, the value of which will be determined using a_{ji}, depending on the type of customer being serviced. If there are no other requests present when the service of the next customer begins (which is obviously of the

second type), the system turns into state 1, and the probabilities of turning into other states from this one are given by a_{2i}. The probabilities of all other transitions are b_i. The corresponding generating functions of all these probabilities are

$$A_j(z) = \sum_{i=0}^{\infty} a_{ji} z^i, \quad B(z) = \sum_{i=0}^{\infty} b_i z^i, \quad \text{and} \quad C(z) = \sum_{i=0}^{\infty} c_i z^i.$$

Collision disciplines. As far as collisions are considered, three different methods will be applied:

Method I. In the case of a collision, both types of customers are refused.

Method II. In the case of a collision, the first type customers are accepted for service, but the second type ones are refused.

Method III. In the case of a collision, customers of both types are accepted for service, but the ones of the first type are served first. When applying this method, in addition to previously defined transition probabilities, new ones have to be introduced. Let a_{12i} denote the probability of the appearance of i customers of the second type at the service of a first type customer, if the service process started with the simultaneous appearance of customers of both types; the generating function of these probabilities is $A_{12}(z) = \sum_{i=0}^{\infty} a_{12i} z^i$.

Let us now summarise the properties of the system and introduce some notation. Consider a discrete cyclic-waiting system serving two types of customers in which inter-arrival time distributions are geometric with parameters r_j, whereas service times are uniformly distributed in the intervals $[\gamma_j, \delta_j]$ $(j = 1, 2)$, respectively. The service of an entering customer may start immediately on arrival if the server is free, but in case of a busy server or waiting customers, first type customers are refused, and second type customers join the queue. The service of queued customers may start at times differing from their arrival times by multiples of the cycle-time T, which is divided into n equal time slices; these form the units of the geometric and uniform distributions. The states of the corresponding embedded Markov-chain are identified with the number of customers in the system at moments just before starting the service of a customer.

Theorem 1. *The matrix of the transition probabilities of the defined chain has the form:*

$$\begin{pmatrix} c_0 & c_1 & c_2 & c_3 & \dots \\ a_{20} & a_{21} & a_{22} & a_{23} & \dots \\ 0 & b_0 & b_1 & b_2 & \dots \\ 0 & 0 & b_0 & b_1 & \dots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{pmatrix}. \tag{1}$$

The elements of the matrix are determined by their generating functions below.

$$A_j(z) = \frac{q_j}{r_2}\left[(1-r_2)^{\frac{\gamma_j}{\gamma}n+1} - (1-r_2)^{\frac{\delta_j}{\gamma}n+1}\right] +$$
$$+ zq_j\left[(1-r_2)^{\frac{\gamma_j}{\gamma}n} - (1-r_2)^{\frac{\delta_j}{\gamma}n}\right] + zq_j\left[(1-r_2)^{\frac{\gamma_j}{\gamma}n} - (1-r_2)^{\frac{\delta_j}{\gamma}n}\right] \times$$
$$\times \frac{1-(1-r_2)^n}{r_2(1-r_2)^n}(1-r_2+r_2z)^n \frac{1-\left(\frac{1-r_2+r_2z}{1-r_2}\right)^{\frac{\gamma_j}{\gamma}n}}{1-\left(\frac{1-r_2+r_2z}{1-r_2}\right)^n} +$$
$$+ zq_j(1-r_2+r_2z)^n\left[n\frac{(1-r_2+r_2z)^{\frac{\gamma_j}{\gamma}n} - (1-r_2+r_2z)^{\frac{\delta_j}{\gamma}n}}{1-(1-r_2+r_2z)^n} -\right.$$
$$\left. - (1-r_2)^{\frac{\delta_j}{\gamma}n}\frac{1-(1-r_2)^n}{r_2(1-r_2)^n}\frac{\left(\frac{1-r_2+r_2z}{1-r_2}\right)^{\frac{\gamma_j}{\gamma}n} - \left(\frac{1-r_2+r_2z}{1-r_2}\right)^{\frac{\delta_j}{\gamma}n}}{1-\left(\frac{1-r_2+r_2z}{1-r_2}\right)^n}\right], \quad (2)$$

$$A_{12}(z) = znq_1\frac{(1-r_2+r_2z)^{\left(\frac{\gamma_1}{\gamma}+1\right)n} - (1-r_2+r_2z)^{\left(\frac{\delta_1}{\gamma}+1\right)n}}{1-(1-r_2+r_2z)^n}, \quad (3)$$

$$B(z) = \frac{r_2q_2}{1-(1-r_2^n)}\frac{(1-r_2+r_2z)^{\frac{\gamma_2}{\gamma}n} - (1-r_2+r_2z)^{\frac{\delta_2}{\gamma}n}}{1-(1-r_2+r_2z)^n}\times$$
$$\times\left[\left((1-r_2+r_2z) - (1-r_2+r_2z)^{n+1}\right)\times\right.$$
$$\times\left(\frac{1-(1-r_2)^n(1-r_2+r_2z)^n}{(1-(1-r_2)(1-r_2+r_2z))^2} - \frac{n(1-r_2)^n(1-r_2+r_2z)^n}{1-(1-r_2)(1-r_2+r_2z)}\right) +$$
$$\left. + n(1-r_2+r_2z)^{n+1}\frac{1-(1-r_2)^n(1-r_2+r_2z)^n}{1-(1-r_2)(1-r_2+r_2z)}\right] \quad (4)$$

and $C(z)$ depends on collision policies:

I. $C(z) = \frac{r_1(1-r_2)}{r_1+r_2-r_1r_2}A_1(z) + \frac{r_2(1-r_1)}{r_1+r_2-r_1r_2}A_2(z) + \frac{r_1r_2}{r_1+r_2-r_1r_2}$, (5a)

II. $C(z) = \frac{r_1}{r_1+r_2-r_1r_2}A_1(z) + \frac{r_2(1-r_1)}{r_1+r_2-r_1r_2}A_2(z)$, (5b)

III. $C(z) = \frac{r_1(1-r_2)}{r_1+r_2-r_1r_2}A_1(z) + \frac{r_2(1-r_1)}{r_1+r_2-r_1r_2}A_2(z) + \frac{r_1r_2}{r_1+r_2-r_1r_2}A_{12}(z)$. (5c)

Proof. Because of the definitions, the construction of the matrix of transition probabilities is straightforward. However, we draw the attention of the reader to the fact

that probabilities a_{1i} do not appear in it explicitly, as customers of the first type can only be accepted when the system is free. These probabilities are represented through probabilities c_i.

First we determine a_{ji}. In this case only one customer is present at the beginning of a service (the one whose service is about to start). Time units are of length $\frac{T}{n}$, and all time intervals are measured using this unit. The service time of the actual customer is denoted by u, and the next one appears v time units after the servicing of the first customer started. In order to get a_{ji}, the distribution of $u-v$ must be known. Two separate calculations have to be carried out.

If $0 < l \leq \frac{\gamma_j}{T} n$:

$$P(u-v=l) = \sum_{k=\frac{\gamma_j}{T}n+1}^{\frac{\delta_j}{T}} q_j \left(1-r_2\right)^{k-l-1} r_2 = q_j \left[\left(1-r\right)^{\frac{\gamma_j}{T}n-l} - \left(1-r\right)^{\frac{\delta_j}{T}n-l} \right];$$

and if $\frac{\gamma_j}{T} n < l \leq \frac{\delta_j}{T} n$:

$$P(u-v=l) = \sum_{k=l+1}^{\frac{\delta_j}{T}} q_j \left(1-r_2\right)^{k-l-1} r_2 = q_j \left[1 - \left(1-r\right)^{\frac{\delta_j}{T}n-l} \right].$$

The waiting time can be determined on the basis of these probabilities. If $u-v=0$ (the next customer appears in the time-slice in which the service of the present customer is completed), then the service of the next customer can be started immediately, the waiting time is 0. If $u-v$ is in $\overline{1,n}$[1], then the waiting time is T, i. e. n units; if it is in $\overline{n+1,2n}$, then the waiting time is $2n$; and in general if $u-v$ takes some value from $\overline{(i-1)\,n+1, in}$, then the waiting time of the next customer is in.

The probability that the waiting time of the second customer is in (i. e. $u-v$ is in $\overline{(i-1)\,n+1, in}$) is the following.

If $0 < i \leq \frac{\gamma_j}{T}$:

$$P\big((i-1)\,n+1 \leq u-v \leq in\big) = \sum_{l=(i-1)n+1}^{in} q_j \left[\left(1-r_2\right)^{\frac{\gamma_j}{T}n-l} - \left(1-r_2\right)^{\frac{\delta_j}{T}n-l} \right] =$$

$$= q_j \left[\left(1-r_2\right)^{\frac{\gamma_j}{T}n} - \left(1-r_2\right)^{\frac{\delta_j}{T}n} \right] \frac{1-\left(1-r_2\right)^n}{r_2\left(1-r_2\right)^n} \frac{1}{\left(1-r_2\right)^{(i-1)n}}.$$

The generating function of the number of customers appearing in a time slice is $1-r_2+r_2 z$, hence, the generating function of the number of customers entering

[1] Notation $\overline{a,b}$ is used for integer intervals, i. e. $\overline{a,b} = [a,b] \cap \mathbb{Z}$.

during the waiting time is:

$$\sum_{i=1}^{\frac{\gamma_j}{T}} q_j \left[(1-r_2)^{\frac{\gamma_j}{T}n} - (1-r_2)^{\frac{\delta_j}{T}n} \right] \frac{1-(1-r_2)^n}{r_2(1-r_2)^n} \frac{(1-r_2+r_2z)^{in}}{(1-r_2)^{(i-1)n}} =$$

$$= q_j \left[(1-r_2)^{\frac{\gamma_j}{T}n} - (1-r_2)^{\frac{\delta_j}{T}n} \right] \times$$

$$\times \frac{1-(1-r_2)^n}{r_2(1-r_2)^n} (1-r_2+r_2z)^n \frac{1-\left(\frac{1-r_2+r_2z}{1-r_2}\right)^{\frac{\gamma_j}{T}n}}{1-\left(\frac{1-r_2+r_2z}{1-r_2}\right)^n}. \quad (6)$$

If $\frac{\gamma_j}{T} < i \leq \frac{\delta_j}{T}$:

$$P\big((i-1)n+1 \leq u-v \leq in\big) = \sum_{l=(i-1)n+1}^{in} q_j \left[1-(1-r_2)^{\frac{\delta_j}{T}n-l} \right] =$$

$$= q_j n - q_j (1-r_2)^{\frac{\delta_j}{T}n} \frac{1-(1-r_2)^n}{r_2(1-r_2)^n} \frac{1}{(1-r_2)^{(i-1)n}},$$

and the generating function of entering customers is:

$$\sum_{i=\frac{\gamma_j}{T}+1}^{\frac{\delta_j}{T}} \left[q_j n - q_j (1-r_2)^{\frac{\delta_j}{T}n} \frac{1-(1-r_2)^n}{r_2(1-r_2)^n} \frac{1}{(1-r_2)^{(i-1)n}} \right] (1-r_2+r_2z)^{in} =$$

$$= q_j n (1-r_2+r_2z)^n \frac{(1-r_2+r_2z)^{\frac{\gamma_j}{T}n} - (1-r_2+r_2z)^{\frac{\delta_j}{T}n}}{1-(1-r_2+r_2z)^n} -$$

$$- q_j (1-r_2)^{\frac{\delta_j}{T}n} \frac{1-(1-r_2)^n}{r_2(1-r_2)^n} (1-r_2+r_2z)^n \frac{\left(\frac{1-r_2+r_2z}{1-r_2}\right)^{\frac{\gamma_j}{T}n} - \left(\frac{1-r_2+r_2z}{1-r_2}\right)^{\frac{\delta_j}{T}n}}{1-\left(\frac{1-r_2+r_2z}{1-r_2}\right)^n}. \quad (7)$$

The probability that the waiting time is zero (which happens when the next customer enters during the last time-slice of the service of the previous one) is:

$$\sum_{k=\frac{\gamma_j}{T}n+1}^{\frac{\delta_j}{T}n} q_j (1-r_2)^{k-1} r_2 = q_j \left[(1-r_2)^{\frac{\gamma_j}{T}n} - (1-r_2)^{\frac{\delta_j}{T}n} \right], \quad (8)$$

while the probability that there is no entry at all is:

$$\sum_{k=\frac{\gamma_j}{T}n+1}^{\frac{\delta_j}{T}n} q_j (1-r_2)^k = \frac{q_j}{r_2} \left[(1-r_2)^{\frac{\gamma_j}{T}n+1} - (1-r_2)^{\frac{\delta_j}{T}n+1} \right]. \quad (9)$$

Bearing in mind that we examined those possibilities when a customer enters obligatorily, generating functions (6), (7), and (8) have to be multiplied by z, then added to (9), which yields (2).

If the third method of collision treatment is applied, then $A_{12}(z)$ has to be determined. In this case the service starts with the simultaneous arrival of two customers of different types, and the one of the first type is served first. The duration of its service can take any value between $\frac{\gamma_1}{T}n+1$ and $\frac{\delta_1}{T}n$ with equal probability $q_1=\frac{T}{n(\delta_1-\gamma_1)}$, so

$$P\big((i-1)n+1\le u\le in\big)=\sum_{l=(i-1)n+1}^{in}q_1=q_1n$$

for all $\frac{\gamma_1}{T}<i\le\frac{\delta_1}{T}$.

Taking into account that one customer of the second type is already present at the start of the service of the first type customer, the generating function is:

$$A_{12}(z)=z\sum_{i=\frac{\gamma_1}{T}+1}^{\frac{\delta_1}{T}}nq_1\left(1-r_2+r_2z\right)^{in}=$$

$$=znq_1\frac{\left(1-r_2+r_2z\right)^{\left(\frac{\gamma_1}{T}+1\right)n}-\left(1-r_2+r_2z\right)^{\left(\frac{\delta_1}{T}+1\right)n}}{1-\left(1-r_2+r_2z\right)^n},$$

which is identical to (3).

The probability of the appearance of at least one customer of any type in a time-slice is

$$1-(1-r_1)(1-r_2)=r_1+r_2-r_1r_2.$$

The busy period can start with the arrival of a first type customer alone with the probability $\frac{r_1(1-r_2)}{r_1+r_2-r_1r_2}$, with the arrival of a single second type customer with the probability $\frac{r_2(1-r_1)}{r_1+r_2-r_1r_2}$, and with the arrival both customers, with the probability $\frac{r_1r_2}{r_1+r_2-r_1r_2}$. These easily explain (5c) (collision discipline III.). In the case of collision treatment method I., customers of both types are lost if they arrive during the same time-slice, and this may be interpreted as a service of zero length (the system stays in the free state with this probability), which results in the generating function (5a). If collision discipline II. is applied, then the busy period can start with the service of a first type customer with the probability $\frac{r_1}{r_1+r_2-r_1r_2}$ (no matter whether there was a refused customer of the second type at the same time), and starts with the service of a second type customer with the probability $\frac{r_2(1-r_1)}{r_1+r_2-r_1r_2}$, which explains (5b).

Finally, we are going to determine the transition probabilities b_i. In this case, when the service of the actual customer begins, the next one is already present. Let $x=u-\left\lfloor\frac{u-1}{n}\right\rfloor n$, i. e. x is the service time mod n $(1\le x\le n)$, and let y denote the inter-arrival time mod n $(1\le y\le n)$. The time elapsed between the starting

moments of services of these two consecutive customers will be:

$$t_0 = \begin{cases} \left\lfloor \frac{u-1}{n} \right\rfloor n + y, & \text{if } x \le y; \\ \left(\left\lfloor \frac{u-1}{n} \right\rfloor + 1 \right) n + y, & \text{if } x > y. \end{cases}$$

Now, let us fix y, and consider the usual set of integers $\overline{in+1,(i+1)n}$. If the service is completed until $in+y$ (inclusive), then the time in question is $in+y$. The probability of this event is yq_2. If the service finishes later than $in+y$, then the time difference between starting services is $(i+1)n+y$ with probability $(n-y)q_2$. Summation has to be extended over all possible values of service times, therefore, the generating function of the number of entering customers on condition that inter-arrival time mod n is equal to y is:

$$\sum_{i=\frac{\gamma_2}{T}}^{\frac{\delta_2}{T}-1} \left[q_2 y \left(1 - r_2 + r_2 z\right)^{in+y} + q_2 (n-y) \left(1 - r_2 + r_2 z\right)^{(i+1)n+y} \right] =$$

$$= q_2 \left(1 - r_2 + r_2 z\right)^y \left[y + (n-y) \left(1 - r_2 + r_2 z\right)^n \right] \times$$

$$\times \frac{\left(1 - r_2 + r_2 z\right)^{\frac{\gamma_2}{T} n} - \left(1 - r_2 + r_2 z\right)^{\frac{\delta_2}{T} n}}{1 - \left(1 - r_2 + r_2 z\right)^n}.$$

The random variable of y has truncated geometric distribution with probabilities $\frac{(1-r_2)^{k-1} r_2}{1-(1-r_2)^n}$ $(k = 1, 2, \ldots, n)$. The previously calculated sum has to be multiplied by $\frac{(1-r_2)^{y-1} r_2}{1-(1-r_2)^n}$, and summed up for y, from 1 to n. Expanding this sum we finally receive (4).

□

Theorem 2. *The generating function of ergodic distribution of this chain is:*

$$P(z) = \sum_{i=0}^{\infty} p_i z^i = \frac{p_0 \big(zC(z) - B(z) \big) + p_1 z \big(A_2(z) - B(z) \big)}{z - B(z)}, \tag{10}$$

where p_0 and p_1 are the first two probabilities of the equilibrium distribution. They are connected with the relation $p_1 = \frac{1-c_0}{a_{20}} p_0$, and

$$p_0 = \frac{1 - B'(1)}{1 - B'(1) + C'(1) + \frac{1-c_0}{a_{20}} \big(A_2'(1) - B'(1) \big)}, \tag{11}$$

where

$$A_j'(1) = -a_{j0} + \frac{T}{\delta_j - \gamma_j} \left[\left(1 - r_2\right)^{\frac{\gamma_j}{T} n} - \left(1 - r_2\right)^{\frac{\delta_j}{T} n} \right] \frac{(1-r_2)^n}{1 - (1-r_2)^n} +$$

$$+ \frac{n r_2}{T} \frac{\gamma_j + \delta_j + T}{2},$$

$$A'_{12}(1) = 1 + \frac{nr_2}{T}\frac{\gamma_1 + \delta_1 + T}{2},$$
$$B'(1) = \frac{nr_2}{T}\frac{\gamma_2 + \delta_2 + T}{2}, \tag{12}$$

and $C'(1)$ *depends on collision policies:*

I. $C'(1) = \frac{r_1(1-r_2)}{r_1+r_2-r_1r_2}A'_1(1) + \frac{r_2(1-r_1)}{r_1+r_2-r_1r_2}A'_2(1)$,

II. $C'(1) = \frac{r_1}{r_1+r_2-r_1r_2}A'_1(1) + \frac{r_2(1-r_1)}{r_1+r_2-r_1r_2}A'_2(1)$,

III. $C'(1) = \frac{r_1(1-r_2)}{r_1+r_2-r_1r_2}A'_1(1) + \frac{r_2(1-r_1)}{r_1+r_2-r_1r_2}A'_2(1) + \frac{r_1r_2}{r_1+r_2-r_1r_2}A'_{12}(1)$.

Proof. The matrix of transition probabilities has the form (1). Using this we can determine the probabilities of the equilibrium distribution denoted by p_l. They satisfy the equations

$$\begin{pmatrix} p_0 \\ p_1 \\ p_2 \\ p_3 \\ \vdots \end{pmatrix} = \begin{pmatrix} c_0 & c_1 & c_2 & c_3 & \dots \\ a_{20} & a_{21} & a_{22} & a_{23} & \dots \\ 0 & b_0 & b_1 & b_2 & \dots \\ 0 & 0 & b_0 & b_1 & \dots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{pmatrix}^T \begin{pmatrix} p_0 \\ p_1 \\ p_2 \\ p_3 \\ \vdots \end{pmatrix},$$

i. e.

$$p_0 = p_0c_0 + p_1a_{20}, \tag{13a}$$
$$p_l = \sum_{k=2}^{l+1} p_k b_{l-k+1} + p_0c_l + p_1a_{2l} \qquad (l \geq 1), \tag{13b}$$

from which we receive the following expression for the generating function:

$$P(z) = \sum_{l=0}^{\infty} p_l z^l = p_0C(z) + p_1A_2(z) + \sum_{l=1}^{\infty}\sum_{k=2}^{l+1} p_k b_{l-k+1} z^l =$$
$$= p_0C(z) + p_1A_2(z) + \sum_{k=2}^{\infty}\sum_{l=k-1}^{\infty} p_k b_{l-k+1} z^{l-k+1} z^{k-1} =$$
$$= p_0C(z) + p_1A_2(z) + B(z)\left(\frac{P(z)}{z} - \frac{p_0}{z} - p_1\right),$$

which yields (10). From (13a)

$$p_1 = \frac{1-c_0}{a_{20}}p_0.$$

To determine p_0, the condition $P(1) = 1$ is used, from which (11) is obtained. □

Lemma 1. *The expression* $C'(1)+\frac{1-c_0}{a_{20}}\left(A_2'(1)-B'(1)\right)$ *is always positive, for any values of the parameters, and for any collision discipline.*

Proof.

$$\frac{1-c_0}{a_{20}}\left(A_2'(1)-B'(1)\right)=$$
$$=(1-c_0)\left[-1+\frac{T}{a_{20}\left(\delta_2-\gamma_2\right)}\left((1-r_2)^{\frac{\gamma_2}{T}n}-(1-r_2)^{\frac{\delta_2}{T}n}\right)\frac{(1-r_2)^n}{1-(1-r_2)^n}\right].$$

Substituting $a_{20}=\frac{T}{n(\delta_2-\gamma_2)}\frac{1-r_2}{r_2}\left((1-r_2)^{\frac{\gamma_2}{T}n}-(1-r_2)^{\frac{\delta_2}{T}n}\right)$ in the formula, we get:

$$\frac{1-c_0}{a_{20}}\left(A_2'(1)-B'(1)\right)=(1-c_0)\left(-1+\frac{nr_2}{1-r_2}\cdot\frac{(1-r_2)^n}{1-(1-r_2)^n}\right).$$

Next, $C'(1)$ is transformed in the following way:

$$C'(1)=\sum_{i=0}^{\infty}ic_i=\sum_{i=0}^{\infty}c_i-c_0+\sum_{i=2}^{\infty}(i-1)\,c_i=1-c_0+\sum_{i=2}^{\infty}(i-1)\,c_i.$$

Substituting all in, we obtain:

$$C'(1)+\frac{1-c_0}{a_{20}}\left(A_2'(1)-B'(1)\right)=\sum_{i=2}^{\infty}(i-1)\,c_i+(1-c_0)\,\frac{nr_2}{1-r_2}\,\frac{(1-r_2)^n}{1-(1-r_2)^n}.$$

From the formula rewritten in this form, it is obvious that

$$C'(1)+\frac{1-c_0}{a_{20}}\left(A_2'(1)-B'(1)\right)>0.$$

□

Theorem 3. *The condition of the existence of ergodic distribution is the fulfilment of the following inequality:*

$$\frac{nr_2}{T}\,\frac{\gamma_2+\delta_2+T}{2}<1. \tag{14}$$

Proof. As the embedded Markov-chain is irreducible and aperiodic, the condition of the existence of ergodic distribution is $0<p_0<1$. Applying Theorem 2 and Lemma 1, $p_0=\frac{1-B'(1)}{1-B'(1)+K}$, where K is a positive constant. Thus, the condition simplifies into

$$1-B'(1)>0,$$

which – together with (12) – gives (14). □

3 Conclusions

We investigated a special queuing system which serves two types of customers: customers of the first type are accepted for service only if the system is free; customers of the second type – if not serviced immediately – join a queue and can start their service after a multiple of the cycle-time has elapsed. Inter-arrival time distributions were geometric, service times were uniformly distributed; three collision treatment methods were considered.

By applying exact methods we gave formulae for transition probabilities and established the condition of ergodicity (14). One remarkable thing about (14) is that it does not depend on the customers of the first type, i. e. such customers have no effect on the ergodicity of the process. Moreover, the formula expressing the condition has a clear probabilistic interpretation. Considering that $\frac{T}{n}\frac{1}{r_2}$ is the average inter-arrival time, the condition rewritten in the form

$$\frac{\gamma_2 + \delta_2}{2} + \frac{T}{2} < \frac{T}{nr_2}$$

expresses the constraint that the sum of the average service and average idle times (on average $\frac{T}{2}$ time is needed for the next customer in the queue to reach the starting position) should be less than the average inter-arrival time.

Although the model was motivated by a real problem, it certainly is only a simplified version of it, which affects its applicability. For instance, it is assumed, without any statistical investigation of real data, that arriving entities form Poisson-processes. Presumably, even if they really do so, the Poisson-processes cannot be homogeneous, almost certainly. The FCFS rule is often broken in real life, too; normally the plane reaching the starting position first is to commence landing. Nevertheless, this simplified model provides exciting tasks to solve, and it can be modified later to fit the real case more precisely.

References

[1] Falin, G.I. and Templeton, J.G.C. *Retrial Queues.* Monographs on Statistics and Applied Probability **75** Chapman & Hall, 1997.

[2] Farkas, G. and Kárász, P. Investigation of a discrete cyclic-waiting problem by simulation. *Acta Acad. Paed. Agriensis, Sectio Mathematicae*, 27:57–62, 2000.

[3] Gnedenko, B.V. and Kovalenko, I.N. *Introduction to Queuing Theory.* Birkhäuser, Boston, 1989.

[4] Kárász, P. Special retrial systems with exponentially and uniformly distributed service times. In proc. of *International Conference in Memoriam John von Neumann*, pages 199–207, Budapest, 2003.

[5] Kárász, P. Special retrial systems with requests of two types. *Theory of Stochastic Processes*, 10(26)3-4:51–56, 2004.

[6] Kárász, P. and Farkas, G. Exact solution for a two-type customers retrial system. *Computers and Mathematics with Applications*, 49:95–102, 2005.

[7] Kárász, P. A special discrete cyclic-waiting queuing system. Submitted to *Central European Journal of Operations Research*, 2008.

[8] Laevens, K., Van Houdt, B., Blondia, C., and Bruneel, H. On the sustainable load of fiber delay line buffers. *Electronic Letters*, 40:137–138, 2004.

[9] Lakatos, L. On a simple continuous cyclic-waiting problem. *Annales Univ. Sci. Budapest., Sect. Comp.*, 14:105–113, 1994.

[10] Lakatos, L. On a cyclic-waiting queuing system. *Theory of Stochastic Processes*, 2(18)1-2:176–180, 1996.

[11] Lakatos, L. On a simple discrete cyclic-waiting queuing problem. *J. Math. Sci. (New York)*, 92(4):4031–4034, 1998.

[12] Lakatos, L. and Koltai, T. A discrete retrial system with uniformly distributed service time. *Annales Univ. Sci. Budapest., Sect. Comp.*, 22:225–234, 2003.

[13] Lakatos, L. and Zbăganu, G. Waiting time in cyclic-waiting systems. *Annales Univ. Sci. Budapest., Sect. Comp.*, 27:217–228, 2007.

[14] Rogiest, W., Laevens, K, Walraevens, J., and Bruneel, H. Analyzing a degenerate buffer with general inter-arrival and service times in discrete time. *Queueing Systems*, 56:203–212, 2007.

Regular Papers

Acta Cybernetica 18 (2008) 697–718.

Splitters and Barriers in Open Graphs Having a Perfect Internal Matching*

Miklós Bartha† and Miklós Krész‡

Abstract

A counterpart of Tutte's Theorem and Berge's formula is proved for open graphs with perfect (maximum) internal matchings. Properties of barriers and factor-critical graphs are studied in the new context, and an efficient algorithm is given to find maximal barriers of graphs having a perfect internal matching.

Keywords: graph matchings, splitters, barriers, factor-critical graphs

1 Introduction

The concepts "open graph" and "perfect internal matching" have emerged from the study of soliton automata introduced in [11]. In this graph theoretical model for electronic switching at the molecular level, an undirected graph represents the topological structure of a hydrocarbon molecule having an alternating pattern of single and double bonds between its carbon atoms. A soliton is a solitary wave that travels through chains of alternating single and double bonds in small packets, and has the ability to switch each affected bond to its opposite. See [10] for the physico-chemical aspects of soliton switching. Soliton waves are initiated and received at some designated interface points, which are treated as distinguished vertices in the graph model. These vertices are called external, and for convenience it is assumed that a vertex is external iff it has degree one. Hence the notion "open graph".

The fact that a molecule has an alternating pattern of single and double bonds is captured by requiring that the underlying graph has a matching (e.g. the collection of double bonds) which covers each vertex, with the possible exception of the external ones. The status of any particular external vertex being covered or not by such a "perfect internal" matching changes when a soliton is initiated from

*Work partially supported by Natural Science and Engineering Research Council of Canada, Discovery Grant #170493-03

†Department of Computer Science, Memorial University of Newfoundland, St. John's, NL, Canada, E-mail: bartha@cs.mun.ca

‡Department of Computer Science, University of Szeged, Faculty of Juhász Gyula Teacher Training College, 6725, Szeged, Boldogasszony sgt. 6, Hungary, E-mail: kresz@jgypk.u-szeged.hu

or received at that vertex. A soliton graph is defined as an open graph having a perfect internal matching. The automaton behavior of soliton graphs arises from the switching capability of the soliton. See [11] for the precise definition of soliton automata.

Even though open graphs and perfect internal matchings have been introduced in [2] with the above specific model in mind, there are other meaningful interpretations of these concepts. Consider, for example, a project on which employees of a company must work in couples. The company employs both full-time and other (e.g. part-time) workers, but its preference is to assign as many full-time employees to the project as possible. In addition, a known compatibility relationship among the employees must be respected, which determines the possible couplings for the job. The underlying graph G in this case consists of the employees as vertices and the compatibility relation among them as edges. Vertices corresponding to full-time employees are considered internal, whereas the group of other employees constitutes the set of external vertices of G. The goal is to find a matching M of G that covers a maximum number of internal vertices.

Considering the latter interpretation of open graphs and perfect/maximum internal matchings, a splitter is a collection of full-time workers such that no two of these workers can work together by any optimal coupling. A barrier, on the other hand, is a collection X of full-time workers (internal vertices) such that, when taking out X from the compatibility graph G, the number of odd internal components (i.e., connected components consisting of an odd number of internal vertices) in $G - X$ exceeds the cardinality of X by the deficiency of G, which is the number of full-time workers remaining idle by any optimal coupling. Clearly, for collections of full-time workers, being a barrier is a stronger property than being just a splitter.

Regarding the soliton automaton model, it turns out that an internal vertex (carbon atom) v may belong to a barrier only if v is "positively inaccessible" for the soliton with respect to any state (perfect internal matching) M. By this we mean that every M-alternating path reaching v (if any) starting from an external vertex will arrive at v on an M-negative edge (i.e. single bond). Consequently, whenever the soliton first arrives at v on edge e, it must return to v before quitting, and then leave v on the same edge e in the opposite direction. (See the definition of soliton paths/walks in [11].) It follows that a viable soliton graph, in which every carbon atom can be reached by the soliton in an appropriate state, has a unique maximal barrier, namely the set of its inaccessible vertices.

The present paper is a synthesis of the results obtained in [4], [5], and [6] with a special emphasis on barriers. A new shorter proof is given for Tutte's Theorem for open graphs with perfect internal matchings, and Berge's Formula is proved as a consequence of this theorem. Barriers are studied in the context of a suitable closure operation, which allows for an analysis of perfect internal matchings in open graphs via perfect matchings of their closures. Maximal splitters in open graphs are compared with maximal barriers in closed graphs, and, using a result from [6], an algorithm is worked out to isolate maximal barriers in linear time.

2 Graphs and matchings

By a graph, throughout the paper, we mean a finite undirected graph in the most general sense, with multiple edges and loops allowed. Our notation and terminology follows [14]. For a graph G, $V(G)$ and $E(G)$ will denote the set of vertices and the set of edges of G, respectively. An edge $e = (v_1, v_2)$ in $E(G)$ connects two vertices $v_1, v_2 \in V(G)$, which are called the *endpoints* of e, and e is said to be *incident* with v_1 and v_2. If $v_1 = v_2$, then e is called a *loop* around v_1. Two edges sharing at least one endpoint are said to be *adjacent* in G. A subgraph G' of G is a graph such that $V(G') \subseteq V(G)$ and $E(G') \subseteq E(G)$. If $X \subseteq V(G)$ then $G[X]$ denotes the subgraph of G for which $V(G[X]) = X$ and $E(G[X])$ consists of the edges of G having both endpoints in X. The notation $G - X$ is a shorthand for $G[V(G) - X]$.

For a graph G, the *degree* of a vertex v, denoted $d(v)$, is the number of occurrences of v as an endpoint of some edge in $E(G)$. According to this definition, every loop around v contributes two occurrences to the count. Vertex v is called *external* if $d(v) = 1$, *internal* if $d(v) \geq 2$ and *isolated* if $d(v) = 0$. *External edges* are those that are incident with at least one external vertex, and an *internal edge* is one that is not external. The sets of external and internal vertices of G will be denoted by $\mathrm{Ext}(G)$ and $\mathrm{Int}(G)$, respectively. Graph G is called *open* if $\mathrm{Ext}(G) \neq \emptyset$, otherwise G is *closed*.

A *matching* M of graph G is a subset of $E(G)$ such that no vertex of G occurs more than once as an endpoint of some edge in M. As the endpoints of loops count twice, such edges cannot participate in M. The endpoints of the edges contained in M are said to be *covered* by M. A *perfect* (*maximum*) matching of G is a matching that covers all (respectively, a maximum number of) vertices in G, and a *perfect internal* (*maximum internal*) matching is one that covers all (respectively, a maximum number of) internal vertices in G. In this paper we are primarily interested in perfect internal matchings of graphs.

An edge $e \in E(G)$ is said to be *allowed (mandatory)* if e is contained in some (respectively, all) perfect internal matching(s) of G. *Forbidden edges* are those that are not allowed. We will also use the term *constant edge* to identify an edge that is either forbidden or mandatory. A *mandatory external vertex* is one that is covered by all perfect internal matchings.

An open graph having a perfect internal matching is called a *soliton graph*. Let G be a soliton graph, fixed for the rest of this section, and let M be a perfect internal matching of G. An edge $e \in E(G)$ is said to be *M-positive* (*M-negative*) if $e \in M$ (respectively, $e \notin M$). An *M-alternating path (cycle)* in G is a path (respectively, even-length cycle) stepping on M-positive and M-negative edges in an alternating fashion. Let us agree that, if the matching M is understood or irrelevant in a particular context, then it will not explicitly be indicated in these terms.

An *external alternating path* is one that has an external endpoint. If both endpoints of the path are external, then it is called a *crossing*. An alternating path is *positive (negative)* if it is such at its internal endpoints (if any), meaning that the edges incident with those endpoints are positive (respectively, negative). A *positive*

(negative) alternating fork is a pair of vertex-disjoint positive (respectively, negative) external alternating paths leading to two distinct internal vertices. Although it sounds somewhat confusing, we still say that these two vertices are *connected* by the fork.

An *alternating unit* is either a crossing or an alternating cycle. *Switching* on an alternating unit amounts to changing the sign of each edge along the unit. It is easy to see that the operation of switching creates a new perfect internal matching for G. Moreover, as it was proved in [1], every perfect internal matching of G can be transformed into any other perfect internal matching by switching on a number of pairwise disjoint alternating units. It follows that any edge e of G is not constant iff there exists an alternating unit passing through e with respect to every perfect internal matching of G.

Since in our treatment we are particular about external vertices, we do not want to allow that subgraphs of G possess external vertices other than the ones present in G. Therefore whenever this happens, and an internal vertex v becomes external in a subgraph G' of G, we shall augment G' by a looping edge around v. This augmentation will be understood automatically throughout the paper.

3 Perfect matchings vs. perfect internal matchings

There is an easy way to neutralize the concession that external vertices in open graphs need not be covered by perfect internal matchings, without actually withdrawing this privilege. In any open graph G, attach a loop around each external vertex to obtain a closed graph $\bar{G}$. Since loops cannot be part of any matching, the augmentation $G \mapsto \bar{G}$ will simply cancel the privilege existing in G by turning every external vertex into an internal one. Thus, perfect matchings of G can trivially be recaptured as perfect internal matchings of $\bar{G}$. This observation will allow us to conveniently refer to results on maximum/perfect matchings without leaving the realm of our current framework dealing with maximum/perfect internal matchings, yet preserving the original scope of these results simply by saying that the objects of consideration are closed graphs.

On the other hand, perfect internal matchings, too, can be studied as ordinary perfect matchings by introducing an appropriate closure operation on graphs.

Definition 3.1. The *closure* of graph G is the closed graph G^* for which:

— $V(G^*) = V(G)$ if $|V(G)|$ is even, and
$V(G^*) = V(G) \cup \{c\}$, $c \notin V(G)$ if $|V(G)|$ is odd;
— $E(G^*) = E(G) \cup \{(v_1, v_2) | v_i \in \mathrm{Ext}(G) \cup \{c\}\}$.

Intuitively, G^* is obtained from G by connecting its external vertices with each other in all possible ways. If $|V(G)|$ happens to be odd, then a new vertex c is added to G, and edges are introduced from c to all of the external vertices. The edges of G^* belonging to $E(G^*) - E(G)$ are called *marginal*, and the vertex c is referred to as the *collector*. Edges incident with the collector vertex will be called collector, too.

Notice that, in the specification of $E(G^*)$, it is not required that $v_1 \neq v_2$. Consequently, in G^*, there will be a loop around each external vertex of G. These loops have no specific role if G has at least two external vertices, although their introduction as trivial forbidden edges is harmless. If there is only one external vertex in G, however, the loop is essential to make G^* closed.

Proposition 3.1. *Graph G has a perfect internal matching iff G^* has a perfect matching.*

Proof. If G^* has a perfect matching M^*, then deleting the marginal edges from G^* and M^* will leave G with a perfect internal matching. Conversely, if G has a perfect internal matching M, then it is always possible to extend M to a perfect matching of G^* by matching up the external vertices of G not covered by M in an arbitrary way, using the collector vertex c if necessary. Obviously, the use of c is necessary if and only if $|V(G)|$ is odd. □

Lemma 3.1. *Every M-alternating crossing of G gives rise to an M^*-alternating cycle of G^* by any extension of M to a perfect matching M^*. Conversely, for an arbitrary perfect matching M^* of G^*, every M^*-alternating cycle of G^* containing at least one marginal edge opens up to a number of alternating crosses with respect to the restriction of M^* to $E(G)$ when the marginal edges are deleted from G^*.*

Proof. Straightforward, using the same argument as under Proposition 3.1. □

Corollary 3.1. *For every edge $e \in E(G)$, e is allowed in G iff e is allowed in G^*.*

Proof. Indeed, by Lemma 3.1,
e is allowed in G
iff there exists a M-alternating unit through e in G
iff there exists an M^*-alternating cycle through e in G^*
iff e is allowed in G^*. □

Recall from [14] that a closed graph G is *elementary* if its allowed edges form a connected subgraph. We shall adopt this definition for open graphs with the additional requirement that the allowed edges must *cover all of the external vertices.*

Based on Corollary 3.1, the following statement was proved in [4].

Proposition 3.2. *A connected graph G is elementary iff G^* is elementary.*

In general, the subgraph of G determined by its allowed edges has several connected components, which are called the *elementary components* of G. An elementary component C is *external* if it contains external vertices of G, otherwise C is *internal*. Notice that an elementary component can be as small as a single external vertex of G. Such a component is called *degenerate*, and it is the only exception from the general rule that elementary components are elementary graphs. A degenerate external component is the external endpoint of a forbidden external edge. A *mandatory elementary component* is a single mandatory edge $e \in E(G)$ with a loop around one or both of its endpoints, depending on whether e is external or

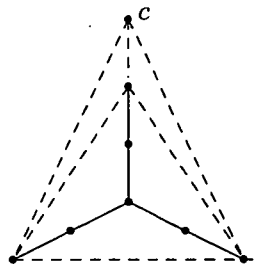


Figure 1: Marginal edges that are forbidden in G^*.

internal. Note that an edge connecting two external vertices is not mandatory in G, therefore it is not a mandatory elementary component either.

Observe that if v is a non-mandatory external vertex and the collector vertex c is present in G^*, then the edge (v, c) cannot be forbidden in G^*. For, if M is a perfect internal matching of G not covering v, then it is always possible to extend M to a perfect matching of G^* by adding the edge (v, c) first. Consequently, if G is elementary, then only those marginal edges can become forbidden in G^* that are different from the collector ones. (An elementary graph G contains a mandatory external vertex iff G consists of a single edge with a number of loops attached to one of its endpoints, in which case the collector vertex is not present in G^*.) Fig. 1 shows a simple example where all these edges are indeed forbidden.

If G is not elementary, then several of its external elementary components may be amalgamated in G^*. The internal elementary components of G, however, will remain intact in G^*, as every forbidden edge of G is still forbidden in G^*. The mandatory external elementary components of G, too, will remain mandatory in G^*. We claim that the union of all non-mandatory external elementary components of G, together with the collector vertex if that is present, forms one elementary component in G^*, called the *amalgamated* elementary component. Indeed, as we have already seen, every collector edge incident with a non-mandatory external vertex is allowed in G^*. Similarly, if e is an edge in G^* connecting two external vertices of G belonging to different non-mandatory elementary components, then it is always possible to find a perfect internal matching M of G by which the two endpoints of e are not covered. Then M can be extended to a perfect matching M^* of G^* by putting in the edge e first, so ensuring that e becomes allowed in G^*.

The observations of the previous paragraph are summarized in Theorem 3.1 below, which provides a characterization of the elementary decomposition of G^*.

Theorem 3.1. *The set of elementary components of G^* consists of:*

(i) the internal elementary components of G;

(ii) the mandatory external elementary components of G;

(iii) the amalgamated elementary component, which is the union of all non-mandatory external elementary components and the collector vertex.

The closure operation provides a hint toward a new interpretation of our framework dealing with open graphs and perfect/maximum internal matchings. The key observation is that, in our arguments relating perfect internal matchings of G to perfect matchings of G^* and vice versa, we did not essentially use the fact that the external vertices have degree 1. The idea works for *any* set of vertices designated as external in G. The concept arising from this remark is that of a perfect (maximum) matching with a "specified potential defect" explained below.

Let G be a graph, and fix $S \subseteq V(G)$ arbitrarily. A perfect (maximum) S-matching of G is a matching M that covers all (respectively, a maximum number of) vertices in S. Vertices in $V(G) - S$ need not, although they may be covered by M. The set $\mathrm{Spd}(G) = V(G) - S$ is the specified potential defect of such matchings, which takes over the role of $\mathrm{Ext}(G)$ in this setting. Although this generalization appears to be substantial for the first sight, a closer look at the definition reveals that it is merely a technical matter. Attach, to each vertex $v \in \mathrm{Spd}(G)$, a handle consisting of two adjacent edges leading to a new external vertex $\hat{v}$. Furthermore, "protect" the vertices in $V(G)$ with degree one by attaching a loop around them. Let $\hat{G}$ denote the resulting graph. Then the restriction of every perfect (maximum) internal matching $\hat{M}$ of $\hat{G}$ to $V(G)$ is a perfect (maximum) S-matching M of G that covers v iff $\hat{M}$ covers $\hat{v}$. Moreover, the connection $\hat{M} \mapsto M$ is a one-to-one correspondence. Consequently, in the study of S-matchings we can always assume, without essential loss of generality, that the vertices belonging to the specified potential defect have degree 1. In this way all substantial results on internal matchings can be rephrased as results on matchings with a specified potential defect in a straightforward way.

The following sections will show that obtaining results on open graphs with perfect/maximum internal matchings from corresponding classical results on closed graphs is by no means a matter of trivial rephrasing, although the results themselves in most cases come as appropriate rewordings of the original statements.

4 Tutte's Theorem and Berge's Formula

First we restate and prove Tutte's well-known theorem [15] in terms of perfect internal matchings. Let $X \subseteq \mathrm{Int}(G)$ be arbitrary, and consider the (connected) components of $G - X$. Component K is called external or internal depending on whether or not K contains external vertices. An *odd internal* component is an internal one consisting of an odd number of vertices. The number of such components is denoted by $c_o^{\mathrm{in}}(G, X)$. If M is a perfect internal matching of G, then by the term "vertex $x \in X$ is *taken* by component K" – or, equivalently, "K takes x" – with respect to M we mean that x is connected to some vertex in K by an M-positive edge.

Theorem 4.1 (Tutte's Theorem). *A graph G has a perfect internal matching iff $c_o^{\mathrm{in}}(G, X) \leq |X|$, for all $X \subseteq \mathrm{Int}(G)$.*

Proof. The "only if" part of the proof is the well-known counting argument: if G

has a perfect internal matching M, then every odd internal component of $G - X$ must take a vertex from X with respect to M, so that $c_o^{in}(G, X) \leq |X|$. To see the "if" part, consider the closure G^* of G, and prove that G^* has a perfect matching whenever $c_o^{in}(G, Y) \leq |Y|$ holds for all $Y \subseteq \mathrm{Int}(G)$. Then, by Proposition 3.1, G will have a perfect internal matching. Using Tutte's original theorem, it is sufficient to show that $c_o(G^*, X) \leq |X|$ holds in G^* for all $X \subseteq V(G^*)$, where $c_o(G^*, X)$ is the number of odd components in $G^* - X$.

To avoid unnecessary complications caused by the collector vertex being present in G^* we can assume, without loss of generality, that $|V(G)|$ is even. If $|V(G)|$ were odd, then we would rather "duplicate" an arbitrary external edge e of G – that is, introduce a new external vertex with an incident edge adjacent to e – than bother with the collector vertex when taking the closure. This slight modification is equivalent to introducing an extra edge from the collector vertex to the internal endpoint of one external edge, which preserves the number $c_o(G^*, X)$ as well as the correspondence between the perfect internal matchings of G and the perfect matchings of G^* explained in Proposition 3.1.

Let $X = Y \cup \{x_1, x_2, \ldots, x_k\} \subseteq V(G^*)$ be arbitrary such that $Y \subseteq \mathrm{Int}(G)$ and $x_i \in \mathrm{Ext}(G)$, $1 \leq i \leq k$ for some $k \geq 0$. We say that a component K in $G^* - X$ is owned by x_i if x_i is connected to an internal vertex of K. Component K is called the joint external component, denoted J_X, if K contains an external vertex of G. Clearly, J_X is unique, provided that $k < |\mathrm{Ext}(G)|$. Observe that each odd component K of $G^* - X$ falls in exactly one of the following three pairwise disjoint groups.

Group g_1: the odd internal components of $G - Y$;
Group g_2: the components owned by the vertices $x_1, \ldots, x_k$;
Group g_3: the component J_X by itself, if it exists and is not in group g_2.

Obviously, $|g_3| \leq 1$, $|g_2| \leq k$, and by assumption, $|g_1| \leq |Y|$. Thus,

$$c_o(G^*, X) \leq |Y| + k + 1 = |X| + 1.$$

It remains to show that $c_o(G^*, X) = |X| + 1$ is impossible. This follows from the fact that the parity of $c_o(G^*, X)$ is the same as that of $|X|$. Indeed, on the one hand, the parity of $|V(G^*)|$ is even. On the other hand, $|V(G^*)| = |X| + |V(G^* - X)|$. Concerning $|V(G^* - X)|$, an odd (even) number of odd components in $G^* - X$ contain an odd (even) number of vertices altogether, and any number of even components contribute an even number of vertices to the count. Thus, in order for $|X|$ and $|V(G^* - X)|$ to have the same parity it is necessary that $|X|$ and $c_o(G^*, X)$ have the same parity, too. This concludes the proof of Theorem 4.1. □

We are going to use Tutte's Theorem to derive Berge's Formula [9] on the deficiency of graphs in our framework. Recall that the *deficiency* of a closed graph G, denoted $\mathrm{def}(G)$, is the number of vertices left uncovered by any maximum matching of G. Then, according to Berge's Formula:

$$\mathrm{def}(G) = \max\{c_o(G, X) - |X| \mid X \subseteq V(G)\},$$

where $c_o(G,X)$ is the number of odd components in $G-X$.

For any graph G, let idef(G) denote the *internal deficiency* of G, that is, the number of internal vertices left uncovered by any maximum internal matching of G.

Theorem 4.2 (The Berge Formula). *For any graph G,*

$$\text{idef}(G) = \max\{c_o^{\text{in}}(G,X) - |X| \mid X \subseteq \text{Int}(G)\}.$$

Proof. If G is closed, then the statement is equivalent to Berge's original formula. Therefore we can assume that G is open. We shall follow the idea outlined in [14, Exercise 3.1.16]. Letting

$$\delta'(G) = \max\{c_o^{\text{in}}(G,X) - |X| \mid X \subseteq \text{Int}(G)\},$$

the inequality $\delta'(G) \leq \text{idef}(G)$ is easily obtained by the standard counting argument seen already in the "only if" part of the proof of Tutte's Theorem. The argument is as follows. If M is any maximum internal matching, then at most $|X|$ odd internal components of $G-X$ can take a vertex from X with respect to M. It is therefore inevitable that at least $c_o^{\text{in}}(G,X) - |X|$ internal vertices of G remain uncovered by M.

Now we turn to proving the inequality $\text{idef}(G) \leq \delta'(G)$. If $\delta'(G) = 0$, then the inequality follows from Theorem 4.1. Assuming that $\delta'(G) \geq 1$, construct a new graph G' from G by adjoining a set H of $\delta'(G)$ new vertices to G, joining each of these vertices to each internal vertex of G and also to each other. Furthermore, attach a loop around each vertex in H to ensure that these vertices become internal in G'. It is sufficient to prove that G' has a perfect internal matching. Indeed, if M' is a perfect internal matching of G', then leaving out those edges of M' which are incident with vertices in H results in a matching M of G that covers at least $|\text{Int}(G)| - \delta'(G)$ vertices, showing that $\text{idef}(G) \leq \delta'(G)$.

We use Theorem 4.1 to show that G' has a perfect internal matching. Let $X \subseteq \text{Int}(G')$ be arbitrary, and concentrate on the set of components in $G'-X$. If $X = \emptyset$, then this set consists of a single external component. (Remember that G is open, and $H \neq \emptyset$.) Thus, $c_o^{\text{in}}(G',X) = 0$. If $|X| \geq 1$ and $H \not\subseteq X$, then $G'-X$ has at most one internal component, so that $c_o^{\text{in}}(G',X) \leq 1 \leq |X|$. If, however, $X = H \cup Y$, then the components of $G'-X$ coincide with those of $G-Y$. Thus,

$$c_o^{\text{in}}(G',X) = c_o^{\text{in}}(G,Y) \leq \delta'(G) + |Y| = |H| + |Y| = |X|.$$

The statement now follows from Tutte's Theorem. □

Another fundamental theorem in matching theory is the Gallai-Edmonds Structure Theorem ([12],[13]). The main idea of this theorem is to decompose a closed graph G into three sets of vertices as follows.

- $D(G)$: vertices not covered by at least one maximum matching of G;
- $A(G)$: vertices in $V(G) - D(G)$ adjacent to at least one vertex in $D(G)$;

- $C(G) = V(G) - A(G) - D(G)$.

The five statements of the theorem are listed below. To explain statements (a) and (d), a closed graph G is called *factor-critical* if $G - v$ has a perfect matching for every $v \in V(G)$. In this case, a *near-perfect matching* of G is one that covers all vertices but one. Clearly, every factor-critical graph is connected and has an odd number of vertices.

(a) The components of the subgraph induced by $D(G)$ are factor-critical.

(b) The subgraph induced by $C(G)$ has a perfect matching.

(c) The bipartite graph obtained from G by deleting the vertices of $C(G)$ and the edges spanned by $A(G)$ and by contracting each component of $D(G)$ to a single vertex has positive surplus (as viewed from $A(G)$).

(d) If M is any maximum matching of G, it contains a near-perfect matching of each component of (the graph induced by) $D(G)$, a perfect matching of $C(G)$, and matches all vertices of $A(G)$ with vertices in distinct components of $D(G)$.

(e) $\mathrm{def}(G) = c(D(G)) - |A(G)|$, where $c(D(G))$ denotes the number of components in $G[D(G)]$.

The counterpart of the Gallai-Edmonds Structure Theorem for maximum internal matchings was proved in [3]. Not surprisingly, the difference between the statement of the original theorem and that of its counterpart is of a rewording nature, which can be summarized as follows:

— the set $D(G)$, as well as $A(G)$, is a subset of $\mathrm{Int}(G)$;

— the subgraph induced by $C(G)$, which will contain all the external vertices, has a perfect internal matching;

— in general, the words "perfect matching" and "maximum matching" are replaced by "perfect internal matching" and "maximum internal matching", respectively;

— in statement (e) above, $\mathrm{def}(G)$ is replaced by $\mathrm{idef}(G)$.

In the light of the Gallai-Edmonds Theorem one can easily argue about the deficiency of the graph G^*. In general, it cannot be expected that $\mathrm{def}(G^*) = \mathrm{idef}(G)$ holds. Equation of these two deficiencies could only be guaranteed if the decision whether to add a collector vertex to $V(G)$ or not depended on the number of vertices in $C(G)$ rather than $V(G)$. This follows immediately from statements (d) and (e) above. If the parity of $|V(G)|$ and $|C(G)|$ is the same, then $C(G^*) = (C(G))^*$, so that $\mathrm{def}(G^*) = \mathrm{idef}(G)$. Otherwise $C(G^*) \neq (C(G))^*$ and $\mathrm{def}(G^*) = \mathrm{idef}(G) + 1$, because the closure of G in this case is implemented through an incorrect closure of $C(G)$, and the discrepancy caused by the missing or unjustified collector vertex in the latter closure contributes $+1$ to the overall deficiency.

5 Splitters, barriers, and the canonical partition of elementary graphs

Recall from [14] that a *barrier* of a closed graph G is a set $X \subseteq V(G)$ for which the maximum is reached in Berge's formula. We extend this definition to open graphs in the following natural way.

Definition 5.1. A *barrier* of graph G is a set $X \subseteq \mathrm{Int}(G)$ for which $|X| = c_o^{\mathrm{in}}(G, X) - \mathrm{idef}(G)$.

Let X be a barrier in graph G (open or closed). It is evident that, for every $x \in X$, $X - \{x\}$ is a barrier in $G - x$. Moreover, the (internal) deficiency of $G - x$ is one greater than that of G. Consequently, $X \subseteq C(G) \cup A(G)$, according to the Gallai-Edmonds decomposition of G. As it was proved in [14, Theorem 3.3.15], $A(G)$ is the intersection of all (inclusionwise) maximal barriers in a closed graph G. For the reader's convenience we repeat the leading argument of this proof here, without assuming that G is closed.

Theorem 5.1. *The set $A(G)$ is contained in every maximal barrier of G.*

Proof. Let X be any maximal barrier. We claim that $A(G - X) = \emptyset$. For, if $A(G-X)$ were not empty, then $X \cup A(G-X)$ would be a barrier properly containing X. It is also easy to see that, for any vertex $u \in A(G)$, $A(G-u) = A(G)-\{u\}$, and for any $u \in C(G)$, $A(G-u) \supseteq A(G)$. (See [14, Lemma 3.2.2] for these statements in closed graphs.) Thus, $A(G) \subseteq A(G - X) \cup X$, so that $A(G - X) = \emptyset$ implies $A(G) \subseteq X$. □

Corollary 5.1. *For every maximal barrier X of G, $X = A(G) \cup Y$, where Y is a maximal barrier of $C(G)$.*

Proof. Evident by Theorem 5.1, since $A(G)$ separates $C(G)$ from $D(G)$. □

Our concern in this paper is with maximal barriers of graphs. Therefore, on the basis of Corollary 5.1, we can restrict our attention to graphs having a perfect internal matching. Let G be a graph (open or closed) having a perfect internal matching, fixed for the rest of this paper. For two internal vertices u and v of G, we say that u and v *attract* (*repel*) each other if an extra edge $e = (u, v)$ becomes allowed (respectively, forbidden) in the graph $G + e$. The binary relation of two vertices repelling each other is denoted by $\sim$. The following simple statement was proved in [4].

Lemma 5.1. *Two internal vertices u and v of G attract each other iff u and v can be connected by a positive alternating path or fork with respect to every perfect internal matching of G.*

A vertex $v \in \mathrm{Int}(G)$ is called *accessible* with respect to a perfect internal matching M if there exists a positive external M-alternating path leading to v. It was proved in [4] that a vertex v is accessible with respect to some perfect internal

matching of G iff v is accessible with respect to all perfect internal matchings of G. It is therefore meaningful to say that vertex v is accessible in G without specifying the matching M. Vertex v is *inaccessible* if it is not accessible. An edge $e \in E(G)$ is called *viable* if at least one endpoint of e is accessible. Otherwise e is said to be *impervious*. See also [11] for an equivalent definition of impervious edges.

Definition 5.2. A set $X \subseteq \mathrm{Int}(G)$ is a *splitter* if every two vertices of X repel each other in G. Splitter X is *inaccessible* if all of its vertices are such.

The concept maximal splitter (maximal inaccessible splitter) is meant inclusion-wise. Notice that, in this way, a maximal inaccessible splitter is not necessarily a maximal splitter.

For any set $X \subseteq \mathrm{Int}(G)$, let G_X be the graph obtained from G by connecting, with an extra edge, each vertex in X with all internal vertices of G, provided that this edge does not already exist in G. If $G = G_X$, then we say that G is *X-complete*.

Lemma 5.2. *For every $X \subseteq \mathrm{Int}(G)$, X is a splitter in G iff X is a splitter in G_X.*

Proof. Let G_e be the graph obtained from G by adding just one edge e towards constructing G_X. In order to prove the lemma it is sufficient to show that if X is a splitter in G, then it is one in G_e as well. Assume, to the contrary, that X is a splitter in G, yet, two vertices $x, y \in X$ attract each other in G_e. Let M be any perfect internal matching of G_e that is also a perfect internal matching of G, i.e., one by which the edge e is negative. By Lemma 5.1, x and y can be connected by a positive M-alternating path or fork β. Leaving out the edge e, β splits into several subpaths. Since one endpoint of e is in X, it is inevitable that one of these subpaths becomes a positive M-alternating path connecting two vertices in X, or two of them constitute a positive M-alternating fork connecting two such vertices. Either way, this contradicts X being a splitter in G. □

We claim that any two distinct vertices u, v in a barrier X of G repel each other. Indeed, assume that the edge $e = (u, v)$ is part of some perfect internal matching M of $G+e$. Since $|X| = c_o^{\mathrm{in}}(G, X) = c_o^{\mathrm{in}}(G_e, X)$, at least two odd components of in $G_e - X$ could not take a vertex from X with respect to M; a contradiction. Thus, every barrier is a splitter. The converse of this statement is not true, however, as shown by the graph of Fig. 2. It is also clear by Corollary 3.1 that a set $X \subseteq \mathrm{Int}(G)$ is a splitter in G iff X is a splitter in G^*. As we shall see, splitters are in close relationship with extreme sets of vertices. Recall from [14] that a set of vertices X in a closed graph G is *extreme* if $\mathrm{def}(G - X) = \mathrm{def}(G) + |X|$.

Proposition 5.1. *A set $X \subseteq V(G)$ of a closed graph G having a perfect matching is a splitter iff X is extreme.*

Proof. By Lemma 5.2 we can assume, without loss of generality, that G is X-complete. If X is extreme, then G cannot have a perfect matching containing an edge in $X \times X$. Indeed, if M was such a matching, then the restriction of M to $G - X$ would cover more than $|V(G)| - |X|$ vertices. Thus, X is a splitter.

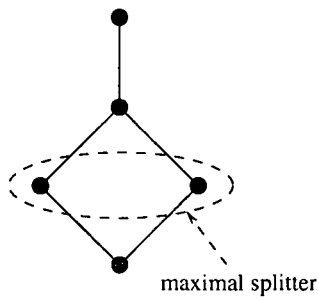


Figure 2: A maximal splitter that is not a barrier

Now let X be a splitter. Obviously, $\mathrm{def}(G - X) \leq |X|$. Assume, by way of contradiction, that $\mathrm{def}(G - X) = |X| - k$ for some $k > 0$, and let M be any maximum matching of $G - X$. Couple up each vertex in $G - X$ not covered by M with an arbitrary vertex in X, and extend M by these edges to form a matching $\bar{M}$ in G. (Remember that G is X-complete.) Observe that k must be odd, otherwise $\bar{M}$ could further be extended to a perfect matching of G containing $k/2$ edges from $X \times X$, contradicting the fact that X is a splitter. On the other hand, k cannot be odd, for $|V(G)|$ is even. □

Proposition 5.2. *A set $X \subseteq \mathrm{Int}(G)$ is a barrier in G^* iff $|X| = c_o^{\mathrm{in}}(G, X)$ or $|X| = c_o^{\mathrm{in}}(G, X) + 1$.*

Proof. Using the trick of duplicating an external edge e of G when $|V(G)|$ is odd, as seen under the proof of Tutte's Theorem, we can assume, without loss of generality, that $|V(G)|$ is even. The reason is that X is a barrier in G^* iff X is one in the closure of the augmented graph G_e. Consider the components in $G - X$ and those in $G^* - X$. The only difference between these two groups is that the external componets in $G - X$ are joined to form one component J_X in $G^* - X$. Assume that J_X is even. Then, clearly, X is a barrier in G^* iff $|X| = c_o^{\mathrm{in}}(G, X)$. On the other hand, if J_X is odd, then X is a barrier in G^* iff $|X| = c_o^{\mathrm{in}}(G, X) + 1$. Moreover, if $|X| = c_o^{\mathrm{in}}(G, X)$, then J_X must be even, and if $|X| = c_o^{\mathrm{in}}(G, X) + 1$, then J_X must be odd. □

Corollary 5.2. *Every barrier of G is also a barrier in G^*.*

Theorem 5.2. *Every maximal splitter of G is a barrier in G^*.*

Proof. We can again assume, without loss of generality, that $|V(G)|$ is even. Let Y be a maximal splitter of G. By Proposition 5.2 it is enough to prove that $|Y| = c_o^{\mathrm{in}}(G, Y)$ or $|Y| = c_o^{\mathrm{in}}(G, Y) + 1$. Since Y is also a splitter in G^*, it is extreme in that graph according to Proposition 5.1. Thus, by [14, Lemma 3.3.8], Y can be extended to a maximal barrier X of G^*. Clearly, $X - Y = \{x_1, \ldots, x_k\} \subseteq \mathrm{Ext}(G)$, because Y is maximal. Concentrate on the odd components of $G^* - X$, and observe

that the situation is analogous to the one analyzed in the proof of Tutte's Theorem. Each of these components falls in one of the three groups specified there. Note that the number of odd components in $G^* - X$ can reach the barrier level $|X| = |Y| + k$ only if the size of group g_1 is at least $|Y| - 1$, that is, $c_o^{in}(G, Y) \geq |Y| - 1$. On the other hand, $c_o^{in}(G, Y) \leq |Y|$ is guaranteed by Tutte's Theorem. The proof is now complete. □

Proposition 5.3. *No barrier exists in an elementary soliton graph G, other than the empty set.*

Proof. The empty set is trivially a barrier in all soliton graphs. By way of contradiction, assume that $X \subseteq V(G)$ is a non-empty barrier. Since $c_o^{in}(G, X) = |X|$, each vertex in X must be taken by an appropriate odd internal component of $G - X$ with respect to any perfect internal matching M of G. Consequently, all edges connecting X to other components of $G - X$ are forbidden. This implies that either the allowed edges of G do not form a connected subgraph, or, when they do, none of the external vertices of G are covered by them; a contradiction. □

Corollary 5.3. *For every maximal splitter X of an elementary soliton graph G having at least one internal vertex, $c_o^{in}(G, X) = |X| - 1$.*

Proof. By Proposition 5.2 and Theorem 5.2 we know that either $c_o^{in}(G, X) = |X| - 1$ or $c_o^{in}(G, X) = |X|$. The latter equation is ruled out, however, due to Proposition 5.3. □

The proof of the following statement uses the exact same argument that was introduced under Proposition 5.3.

Proposition 5.4. *Every barrier X of G is an inaccessible splitter.*

Proof. As it has been noticed earlier, every barrier is a splitter. It is therefore sufficient to prove that every vertex of X is inaccessible. Let M be an arbitrary perfect internal matching of G. Since X is a barrier, every vertex $v \in X$ is taken by some odd internal component of $G - X$ with respect to M. Consequently, any alternating path starting out from v on an M-positive edge is locked forever inside the subgraph of G determined by the odd internal components of $G - X$ plus X. In other words, v is inaccessible. □

It is well-known (cf. [14]) that the collection of maximal barriers in a closed elementary graph G forms a partition of $V(G)$, called the *canonical partition* of G. Canonical partition is established in open graphs in the same way, using maximal splitters rather than barriers.

Theorem 5.3. *The collection of maximal splitters in an elementary graph G forms a partition of* $\mathrm{Int}(G)$.

Proof. Let $\mathcal{P} = \{X_1, \ldots, X_n\}$ be the collection of maximal splitters in G. By Theorem 5.2, each X_i $(1 \leq i \leq n)$ can be extended to a maximal barrier X_i^* of G^*. Since $X_i^* \setminus X_i$ may only contain external vertices of G for every $1 \leq i \leq n$, it follows that $\mathcal{P}$ is the restriction of the canonical partition of G^* to $\mathrm{Int}(G)$. Thus, $\mathcal{P}$ is a partition itself. □

Theorem 5.3 above states that the relation $\sim$ of two internal vertices repelling each other is an equivalence of $\mathrm{Int}(G)$, provided that G is elementary. This fact was first observed in [1]. If G is not elementary, then $\sim$ fails to be transitive in general. It is an important question, however, if the restriction of $\sim$ to a concrete non-degenerate elementary component C of G, denoted $\sim|_C$, is still an equivalence, and if so, does it coincide with canonical equivalence in C alone? As it was pointed out in [4], $\sim|_C$ can be specified as canonical equivalence in the elementary graph C_h, which is obtained from C by adding the so called "hidden edges". A hidden edge (u, v) in C between two distinct internal vertices arises from a negative alternating path or fork α connecting u and v with respect to any perfect internal matching M of G, such that no vertex of α, other than its two endpoints u and v, lies in C. Following [14], if α is a path, then it is called a *negative (M-)ear* to C. Clearly, all hidden edges are forbidden both in G and C_h, but their presence affects canonical equivalence in C_h in such a way that it will eventually coincide with $\sim|_C$. See Fig. 3 for two hidden edges, one in elementary component C_1, and the other in C_4. Notice that the two vertices in C_1 not connected by the hidden edge fall in the same canonical class in C_1, but different canonical classes according to $(C_1)_h$. This holds for the elementary component C_4 as well. Let us agree that, in the future, by a canonical class of C we shall in fact mean one of C_h.

6 Finer structure of maximal splitters and barriers

Recall that a closed graph G is *factor-critical* if $G - v$ has a perfect matching for every $v \in V(G)$. We shall adopt this definition word by word for open graphs, assuming of course that $v \in \mathrm{Int}(G)$, and requiring that $G - v$ has a perfect internal matching. We also require that G be connected, because, unlike for closed graphs, this property does not come as a consequence. The following simple result is quoted from [5].

Proposition 6.1. *A connected open graph G is factor-critical iff G has a perfect internal matching and every internal vertex in G is accessible.*

Corollary 6.1. *No barrier exists in factor-critical open graphs, other than the empty set.*

Proof. Immediate by Propositions 5.4 and 6.1. Notice that the statement trivially holds for closed graphs as well, even though such graphs do not have a perfect (internal) matching. The reason is that a factor-critical closed graph G is the

single component of $D(G)$ by itself, and all barriers lie completely in $A(G) \cup C(G)$. (See the Gallai-Edmonds decomposition of graphs.) □

Using factor-critical graphs, the following characterization of maximal splitters was obtained in [5]. Recall that a component K is *degenerate* if K consists of a single external vertex of G.

Theorem 6.1. *For a set X of internal vertices of a soliton graph G, the following two statements are equivalent.*

(i) The set X is a maximal splitter.

(ii) Each non-degenerate component of $G - X$ is factor-critical such that

(iia) $|X| = c_o^{in}(G, X) + 1$, or

(iib) $|X| = c_o^{in}(G, X)$ with every external component of $G - X$ being degenerate.

Furthermore, condition (iib) holds in (ii) above iff X is inaccessible.

The structure of elementary components in a soliton graph G has been analysed in [4]. To summarize the main results of this analysis, we first need to review some of the key concepts introduced in that paper. The reader can obtain a good understanding of these concepts by following the definitions to come on Fig. 3.

An elementary component of G is *viable* if it does not contain impervious allowed edges. (Recall that an edge e is impervious if both endpoints of e are inaccessible.) In Fig. 3, all elementary components, with the exception of C_7, are viable. A viable internal elementary component C is *one-way* if all external alternating paths (with respect to any perfect internal matching M) enter C in vertices belonging to the same canonical class of C. This unique class, as well as the vertices belonging to this class, are called *principal*. Furthermore, every non-degenerate external elementary component is considered a priori one-way (with no principal canonical class, of course). In Fig. 3, elementary components C_1, C_4, and C_6 are one-way internal, with their principal vertices encircled. A viable elementary component is *two-way* if it is not one-way. An *impervious* elementary component is one that is not viable.

We say that elementary component C' is *two-way accessible* from component C with respect to any (or all) perfect internal matching(s) M, in notation $C \rho C'$, if C' is covered by a negative (M-)ear to C. The ear itself might be closed, meaning that its two endpoints are the same. It is required, though, that if C is one-way and internal, then the endpoints of this ear *are not* in the principal canonical class of C. As it was shown in [4], the two-way accessible relationship is matching invariant. In Fig. 3, C_2 is two-way accessible from C_1, C_3 from C_2, and C_5 from C_4. (But C_3 is not two-way accessible from C_1, and C_2, C_3, C_4, C_5 are not two-way accessible from C_6, even though there exists a negative closed ear originating from the principal vertex of C_6 that covers all four of these components.) It was also proved in [4] that the transitive closure of the two-way accessible relationship between elementary components is asymmetric.

A *family* of elementary components in G is a block of the partition induced by the smallest equivalence relation containing ρ. A family $\mathcal{F}$ is called external if it contains an external elementary component, otherwise $\mathcal{F}$ is internal. Family $\mathcal{F}$

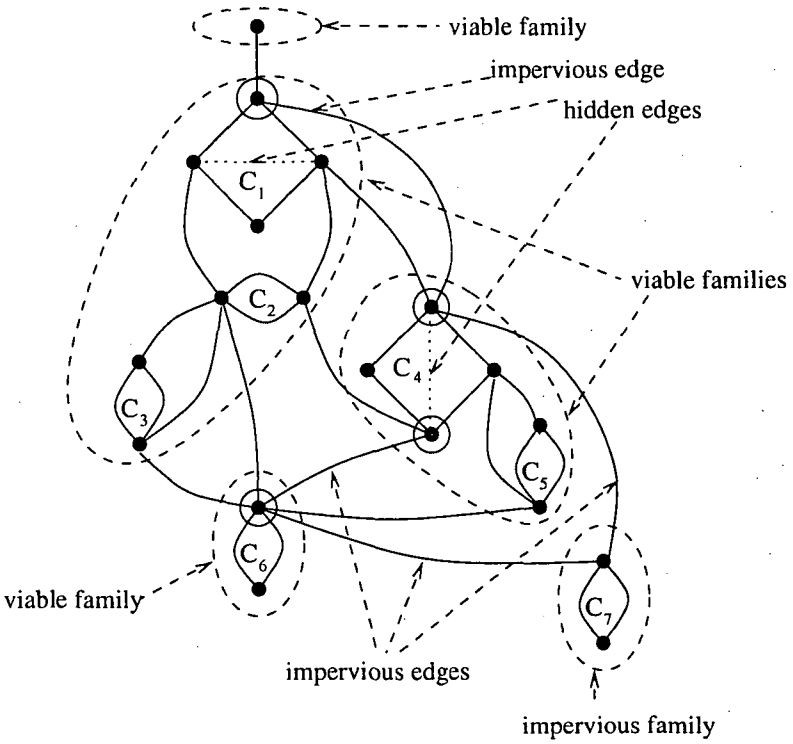


Figure 3: The structure of elementary components in a soliton graph

is *viable* if every elementary component in $\mathcal{F}$ is such. Otherwise the family $\mathcal{F}$ is impervious. A soliton graph G is *viable* if all of its families are such. The graph of Fig. 3 has five families, four of which are viable. The only external family is a stand-alone degenerate external elementary component.

The first group of results obtained in [4] on the structure of elementary components of G can now be stated as follows.

Theorem 6.2. *Each viable family of G contains a unique one-way elementary component, called the root of the family. Each internal vertex in every member of the family, except for the principal vertices of the root, is accessible. The principal vertices themselves are inaccessible, but all other vertices are only accessible through them.*

For two distinct viable families $\mathcal{F}_1$ and $\mathcal{F}_2$, $\mathcal{F}_2$ is said to *follow* $\mathcal{F}_1$, in notation $\mathcal{F}_1 \mapsto \mathcal{F}_2$, if there exists an edge in G connecting any non-principal vertex in $\mathcal{F}_1$ with a principal vertex of the root of $\mathcal{F}_2$. The reflexive and transitive closure of $\mapsto$ is denoted by $\overset{*}{\mapsto}$. The second group of results in [4] characterizes the edge-connections between members inside one viable family, and those between two different families.

Theorem 6.3. *The following three statements hold for the families of any soliton graph G.*

1. *An edge e inside a viable family $\mathcal{F}$ is impervious iff both endpoints of e are in the principal canonical class of the root. Every forbidden edge e connecting two different elementary components in $\mathcal{F}$ is part of a negative ear to some member $C \in \mathcal{F}$.*
2. *For every edge e connecting a viable family $\mathcal{F}_1$ to any other family (viable or not) $\mathcal{F}_2$, at least one endpoint of e is principal in $\mathcal{F}_1$ or $\mathcal{F}_2$. If the endpoint of e in $\mathcal{F}_1$ is not principal, then $\mathcal{F}_2$ is viable and it follows $\mathcal{F}_1$.*
3. *The relation $\overset{*}{\mapsto}$ is a partial order between viable families, by which the external families are maximal elements. This relation reflects the order in which families are reachable by alternating paths starting from external vertices.*

In the light of Theorem 6.2 and Proposition 6.1 it is immediate that a soliton graph G is factor-critical iff G consists of a single non-degenerate external family. Thus, Corollary 6.1 is in fact a generalization of Proposition 5.3. The following theorem is a further generalization along this line.

Theorem 6.4. *Every viable soliton graph has a unique maximal barrier, which is the collection of its inaccessible vertices.*

Proof. By Theorem 6.2 and Proposition 5.4 it is sufficient to show that the set P of all principal vertices of a viable soliton graph G is a barrier. Let $\mathcal{F}$ be an arbitrary internal family of G with root C, and let X_C be the the set of principal vertices in C. By Theorem 6.3, the principal vertices of the families that follow $\mathcal{F}$ separate $\mathcal{F}$ from all the families that are below $\mathcal{F}$ in the Hasse diagram of the partial order $\overset{*}{\mapsto}$. Similarly, the vertices X_C separate all other families from $\mathcal{F}$. Thus, we can

concentrate on the family $\mathcal{F}$ alone as a closed graph, and prove that X_C is a barrier in that graph. Doing this for all internal families will then prove that P is a barrier in G.

As we have already seen, X_C is a canonical class of C_h (remember the extra hidden edges being present in C_h), therefore a maximal barrier in that graph. Let K be an odd component of $\mathcal{F} - X_C$, and consider an arbitrary elementary component D of G present as a subgraph in $K - C$. As D is a two-way member of family $\mathcal{F}$, it can be reached by a cascade of negative ears originating from the root C. Since all hidden edges are present in C_h, and the graph $K - C$ — being essentially a group of interconnected two-way elementary components of $\mathcal{F}$ — has an even number of vertices, the restriction of K to C defines an odd component of $C_h - X_C$. Moreover, this correspondence between the odd components of $\mathcal{F} - X_C$ and those of $C_h - X_C$ is one-to-one. Consequently, since X_C is a barrier in C_h, it must be one in $\mathcal{F}$ as well. □

Corollary 6.2. *Every maximal inaccessible splitter of G is a barrier.*

Proof. Let $v(G)$ be the subgraph of G determined by its viable families. We first prove that each principal vertex u of G repels each internal vertex w lying in $G - v(G)$. Let M be a perfect internal matching of G and suppose, by way of contradiction, that there exists a positive M-alternating path p connecting u and w. Furthermore, starting from u let z denote the last vertex of p which belongs to $v(G)$. It is clear that the subpath of p from u to z is positive at its both ends. However, by Theorem 6.3, z is also a principal vertex, consequently Theorem 6.4 implies that $u \sim z$, which is a contradiction.

By the previous paragraph, every maximal inaccessible splitter X of G is the union of the set S of principal vertices in G and a maximal splitter Y in $G - v(G)$. Since $G - v(G)$ is a closed graph, Y is a barrier in that graph by Theorem 5.2. Also, S is a barrier in $v(G)$. By Theorem 6.3, all edges connecting $v(G)$ to $G - v(G)$ originate from vertices in S, which implies that $X = S \cup Y$ is a barrier in G. □

Corollary 6.3. *A set $X \subseteq \mathrm{Int}(G)$ is a maximal barrier in G iff X is a maximal inaccessible splitter.*

Proof. Immediate by Proposition 5.4 and Corollary 6.2. □

If G is closed, then Corollary 6.3 says that maximal splitters coincide with maximal barriers in G. This result follows already from Theorem 5.3, considering that $G^* = G$ for closed graphs.

On the basis of Corollaries 6.2 and 6.3 we can outline a simple procedure to find one random maximal barrier in a soliton graph G. The procedure uses a global set variable B, the contents of which is initially empty.

Step 1. Isolate the subgraph $v(G)$ consisting of the viable families of G, and add the principal (inaccessible) vertices of $v(G)$ to B.

Step 2. If $G = v(G)$, then terminate. Otherwise, in the remainder graph $G - v(G)$ – wich is now closed – attach an external edge to an arbitrary vertex u to obtain a soliton graph G_u. Set $G := G_u$, and goto *Step 1.*

Clearly, the maximal barriers of G_u coincide with the ones of $G-v(G)$ containing vertex u. (Note that u is trivially inaccessible in G_u.) Therefore the procedure above is capable of finding any maximal barrier of G by chosing the vertex u in *Step 2* in an appropriate way. It was proved in [6] that *Step 1* of the above procedure takes linear time in terms of the number of edges in $v(G)$, provided that a perfect internal matching M has previously been found for G. Also notice that, if G is closed, then choosing a vertex $u \in V(G)$ to be part of a maximal barrier is equivalent to turning G into a soliton graph by attaching an external edge to u before applying the above procedure. Thus, we have proved the following result.

Theorem 6.5. *A random maximal barrier of G (open or closed) can be found in linear time, provided that a perfect internal matching has previously been constructed for G.*

We wish to emphasize that the above procedure can only be used to find a *random* maximal barrier of G in linear time. Finding e.g. a *maximum size* barrier X is a much more complicated issue, which would have to be addressed in a different way. See [7, 8]. Our contribution in this regard concerns only the implications of adding one particular vertex to X.

Let G be a viable soliton graph and X be its maximal barrier. The question arises how X can be extended to a maximal splitter Y of G. By Theorem 6.1 we know that a proper extension exists iff G has non-degenerate external families. Then an obvious way to construct Y is to add an arbitrary maximal splitter of any (one) non-degenerate external family to X. Observe that this is the only way to achieve the goal, since any two internal vertices belonging to different external families attract each other.

Another interesting issue is to relate the maximal barriers of G^* to the maximal splitters of G. Notice that the restriction X of a maximal barrier X^* in G^* to $\mathrm{Int}(G)$ need not be maximal as a splitter in G. For example, if G has a single external family $\mathcal{F}$ with the root of this family being a mandatory edge, then X^* might contain the external endpoint v of this edge (as the only external vertex in G). Since $\mathcal{F}$ is a factor-critical graph, v must be the only vertex from $\mathcal{F}$ present in X^*. Vertex v, however, could be replaced by any other maximal splitter of $\mathcal{F}$ in X^*, so that the result would be a maximal splitter of G as well as a maximal barrier of G^*. Clearly, the restriction X of the original X^* is a maximal barrier of G in this case. Another example of this nature manifests itself when all vertices in X^* belonging to the amalgamated elementary component A of G^* are from $\mathrm{Ext}(G)$, but A does contain vertices in $\mathrm{Int}(G)$. Despite these examples, we still have the following positive result.

Theorem 6.6. *Let X be a maximal splitter in soliton graph G. Then X is either a maximal barrier in G, or it can be extended in a unique way to a maximal barrier X^* of G^*.*

The proof is preceded by a preliminary observation.

Lemma 6.1. *Let u and v be distinct vertices of a closed graph G such that $u \sim v$. Assume, furthermore, that there exists a perfect matching M in G and an M-alternating path p connecting u with v in such a way that p is M-positive at its v end. Then, for an arbitrary vertex z, $z \sim v$ only if $z \sim u$.*

Proof. Suppose, on the contrary, that for some vertex $z \in V(G)$ with $z \sim v$ there exists a positive M-alternating path p' connecting z and u. Starting from v, let w denote the first vertex of p which is also on p'. The prefix of p' from z to w, joined with the section of p from w to v then becomes a path, which cannot be M-alternating, for the positivity of this path would contradict $z \sim v$. But then the section of p from v to w, continued with the suffix of p' from w to u does form a positive M-alternating path, which contradicts $v \sim u$. □

Proof. (of Theorem 6.6) By Theorem 6.1 we can assume that X contains an accesible vertex v. Indeed, if this is not the case, then X is inaccessible, so that $|X| = c_o^{\text{in}}(G, X)$, meaning that X is a barrier in G. We show that X can be extended in a unique way to a barrier of G^*. Clearly, $X^* - X \subseteq \text{Ext}(G)$. Consider the graph G^* as the underlying graph in Lemma 6.1, and notice that for any mandatory external vertex $u \in \text{Ext}(G)$, either $u \not\sim v$, or u and v satisfy the conditions of Lemma 6.1 with a suitable alternating path p that starts out from v on a negative marginal edge. In either case, Lemma 6.1 implies that $u \in X^*$ iff u is present in all maximal barriers of G^* containing v.

Now let $u \in \text{Ext}(G)(\cup\{c\})$ be in the amalgamated elementary component of G^*. It is again true that there exists an alternating path p with respect to some perfect matching of G^* connecting u and v in such a way that p is positive at its v end. This is trivial if v is accessible from u in G, but even if v is accessible from some other external vertex u' in G through path p', this path p' can be augmented by one or two marginal edges to obtain a suitable path p in G^* starting already from u. If $u \not\sim v$, then u cannot be present in any maximal barrier containing v. On the other hand, if $u \sim v$, then Lemma 6.1 applies and $u \in X^*$ iff u is present in all maximal barriers containing v. □

7 Conclusion

We have proved a counterpart of Tutte's Theorem and Berge's Formula for open graphs with perfect (maximum) internal matchings. We have also provided a comparison between barriers in open and closed graphs, and studied the finer structure of maximal splitters and barriers on the basis of earlier results.

An algorithm has been given to find the maximal barriers of an open or closed graph. This algorithm isolates a random maximal barrier in linear time, provided that a perfect internal matching has previously been found for the graph. Maximal splitters of open graphs have been extended to maximal barriers of their closures, and it was proved that this extension is unique, unless the maximal splitter in hand is already a maximal barrier of the original graph.

References

[1] M. Bartha, E. Gombás, On graphs with perfect internal matchings, *Acta Cybernetica* **12** (1995), 111-124.

[2] M. Bartha, E. Gombás, A structure theorem for maximum internal matchings in graphs, *Information Processing Letters* **40** (1991), 289-294.

[3] M. Bartha, The Gallai-Edmonds algebra of graphs, *Congressus Numerantium* **123** (1997), 205–219.

[4] M. Bartha, M. Krész, Structuring the elementary components of graphs having a perfect internal matching, *Theoretical Computer Science* **299** (2003), 179–210.

[5] M. Bartha, M. Krész, Tutte type theorems in graphs having a perfect internal matching, *Information Processing Letters* **91** (2004), 277–284.

[6] M. Bartha, M. Krész, Isolating the families of soliton graphs, *Pure Mathematics and Applications* **13** (2002), 49–62.

[7] D. Bauer, H. J. Broersma, A. Morgana, E. Schmeichel, Tutte sets in graphs I: Maximal tutte sets and D-graphs, *Journal of Graph Theory* **55** (2007), 343–358.

[8] D. Bauer, H.J. Broersma, N. Kahl, A. Morgana, E. Schmeichel and T. Surowiec, Tutte sets in graphs II: The complexity of finding maximum Tutte sets, *Discrete Applied Mathematics* **155** (2007), 1336-1343

[9] C. Berge, Sur le couplage maximum d'un graphe, *C. R. Acad, Sci. Paris Sér. I Math.* **247** (1958), 258–259.

[10] F. L. Carter, Comformational switching at the molecular level, in *Molecular Electronic Devices* (F. L. Carter ed.), Marcel Dekker, Inc., New York, 1982, pp. 51–72.

[11] J. Dassow, H. Jürgensen, Soliton automata, *J. Comput. System Sci.* **40** (1990), 154-181.

[12] J. Edmonds, Paths, trees and flowers, *Canad. J. Math.* **17** (1965), 449–467.

[13] T. Gallai, Maximale Systeme unabhängiger Kanten, *Magyar Tud. Akad. Mat. Kutató Int. Közl.* **9** (1964), 401–413.

[14] L. Lovász, M. D. Plummer, *Matching Theory*, North Holland, Amsterdam, 1986.

[15] W. T. Tutte, The factorization of linear graphs, *J. London Math. Soc.* **22** (1947), 107–111.

Received 10th October 2006

Acta Cybernetica 18 (2008) 719–731.

M-Solid Varieties of Languages

Pedro Baltazar*

Abstract

In this paper, a characterization of the language varieties and congruence varieties corresponding to M-solid pseudovarieties is presented. Taking into account the isomorphisms of the Eilenberg-type correspondences, each complete sublattice of pseudovarieties corresponds to a complete sublattice of language varieties, as well as another one of congruence varieties. For the varieties of tree language, we present the complete sublattices of varieties of languages and the complete sublattice of varieties of congruences isomorphic to the complete sublattice of all M-solid pseudovarieties.

Keywords: tree languages, Eilenberg-type correspondences, M-solid pseudovarieties, M-solid varieties of languages

1 Introduction

Motivated by the connection between star-free languages and aperiodic monoids, and other important similar results, Eilenberg [6] establishes an isomorphism between the lattice of all monoid pseudovarieties and the lattice of all varieties of regular languages. At the beginning of the eighties, Thérien [14] proved that these two lattices are also isomorphic to the lattice of all varieties of congruences of the free monoids. These connections were independently extended to tree languages by Almeida [1] and Steinby [12]. Due to the original result achieved by Eilenberg these kind of connections have come to be known as Eilenberg-type correspondences. Some of the complete sublattices of the complete lattice $\mathcal{L}^{ps}(\tau)$ of all pseudovarieties of type τ were described by Denecke and Pibaljommee in [4]. They showed that for each monoid M of hypersubstitutions, the set $\mathcal{S}_M^{ps}(\tau)$ of all M-solid pseudovarieties of type τ is a complete sublattice of $\mathcal{L}^{ps}(\tau)$. So, it is a natural problem to find a characterization of the complete sublattices corresponding to $\mathcal{S}_M^{ps}(\tau)$, under the Eilenberg-type correspondences. This work is based on the final remarks of Ésik's in [7], where he points out a more wide framework to characterize varieties of tree languages. Following Ésik suggestions we show how monoids of hypersubstitutions and solid pseudovarieties can be used in the characterization of varieties of languages.

*Security and Quantum Information Group, Institute for Telecommunications, Av. Rovisco Pais, 1049-001 Lisboa, Portugal, E-mail: pbtz@math.ist.utl.pt

We assume the reader is familiar with the basic notions and results of Universal Algebra [2]. Throughout this article we fix an algebraic type τ consisting of finitary operations. For technical reasons we will consider a type of algebras without nullary operations. Let $\{f_i : i \in I\}$ be a set of operational symbols of type τ, were f_i is an operational symbol of arity $n_i \geqslant 1$. We will denote by $Alg_f(\tau)$ the class of all finite algebras of type τ. Let $X_\omega = \{x_1, \ldots, x_n, \ldots\}$ be a countable infinite set of variables disjoint from the set of operational symbols, and $X_n = \{x_1, \ldots, x_n\}$ be the set of the first n variables. We will use X to represent any of the previous sets of variables. The set of all n-ary terms of type τ, or terms of type τ over X_n, is denoted by $T_\tau(X_n)$, and by $T_\tau(X_\omega) = \cup_{n \geqslant 1} T_\tau(X_n)$ we denote the set of all terms of type τ. For any term $t \in T_\tau(X)$ we denote by $hg(t)$ the height of the term t. A pseudovariety V of type τ is a class of finite algebras of type τ closed under formation of homomorphic images, subalgebras and finitary direct products. It is well-known that pseudovarieties are defined by filters of equations [2], and that the set $\mathcal{L}^{ps}(\tau)$ of all pseudovarieties of type τ forms a complete lattice. A pseudovariety defined by equations is called an equational pseudovariety. In the sequel, we will consider a non-trivial pseudovariety V of type τ. Let $\mathcal{L}^{ps}(V)$ denote the complete lattice of all subpseudovarieties of V. Given two algebras $\mathbf{A}$ and $\mathbf{B}$, we say that $\mathbf{A}$ *divides* $\mathbf{B}$, and we write $\mathbf{A} \prec \mathbf{B}$, if $\mathbf{A}$ is a homomorphic image of a subalgebra of $\mathbf{B}$. By $Pol_n\mathbf{A}$ we denote the set of all n-ary polynomial operations of the algebra $\mathbf{A}$. In the next two sections we give the necessary definitions and results which will be used to achieve the main results.

2 Eilenberg-type correspondences

The Eilenberg and Thérien results were generalized by Almeida and Steinby using a more general framework that included both cases: varieties of string languages and varieties of tree languages. They considered sets of the finitely generated V-free algebras $\mathbf{F}_nV$ [1]. When V is the pseudovariety of all monoids, the subsets of $\mathbf{F}_nV$ are string languages, and when V is the pseudovariety of all finite algebras of type τ we have the tree languages case.

Let $\mathbf{A}$ be an algebra and $L \subseteq A$ any subset of A. The *syntactic congruence* of L on $\mathbf{A}$ is the relation $\sim_L$ given by

$$a \sim_L b \quad \text{iff} \quad p(a) \in L \Leftrightarrow p(b) \in L,$$

to every unary polynomial operation $p \in Pol_1(\mathbf{A})$, with $a, b \in A$.

The relation $\sim_L$ is the greatest congruence of $\mathbf{A}$ for which L is the union of classes.

The *syntactic algebra* $\mathbf{A}/L$ of the subset L of A is the quotient algebra $\mathbf{A}/\sim_L$, and the homomorphism $\varphi_L : \mathbf{A} \to \mathbf{A}/L$ is called the *syntactic homomorphism* of L. We say that an algebra is syntactic if it is isomorphic to the syntactic algebra of some subset of some algebra.

[1] We follow the somewhat nonstandard definition of V-free algebras from [1] which does not require that a V-free algebra be itself in V.

An operation on subsets of an algebra $\mathbf{A}$ of the form $L \mapsto p^{-1}L := \{a \in A : p(a) \in L\}$, where $p \in Pol_1\mathbf{A}$ is an unary polynomial operation of $\mathbf{A}$, and $L \subseteq A$ is a subset of A, will be called *cancellation*. Another operation on subsets is $L \mapsto \varphi^{-1}L := \varphi^{-1}(L)$ where $\varphi : \mathbf{A} \to \mathbf{B}$ is a homomorphism, and will be referred to as *inverse homomorphism*.

Let $\mathbf{A}$ be an algebra of type τ. A subset $L \subseteq A$ of A is called *V-recognizable* if there exists an algebra $\mathbf{B} \in V$, a homomorphism $\varphi : \mathbf{A} \to \mathbf{B}$, and a subset $K \subseteq B$ such that $L = \varphi^{-1}(K)$. In this case, we say that the triple $\langle \mathbf{B}, \varphi, K \rangle$ recognizes L, or simply that L is recognized by $\mathbf{A}$.

We are only interested in the V-recognizable subsets of the finitely generated V-free algebras $\mathbf{F}_nV$. For any $n \geqslant 1$, let Rec_nV denote the set of all V-recognizable subsets of $\mathbf{F}_nV$. We will refer to the elements of Rec_nV as *V-languages*. By a field of subsets we mean a Boolean subalgebra of the power set, with the usual set operations.

A *variety of V-languages* is a sequence $\mathscr{V} = (\mathscr{V}_n)_{n\geqslant 1}$ such that

L.1) $\mathscr{V}_n$ is a field of subsets of Rec_n;

L.2) $\mathscr{V}_n$ is closed under cancellation;

L.3) $\mathscr{V}$ is closed under inverse homomorphisms $\mathbf{F}_nV \to \mathbf{F}_mV$.

The lattice of all varieties of V-languages is represented by $\mathcal{VL}(V)$.

We represent by Con_nV the set of all congruences θ on $\mathbf{F}_nV$ such that $\mathbf{F}_nV/\theta \in V$ which we will call V-congruences. Let $\varphi : \mathbf{A} \to \mathbf{B}$ be a homomorphism and θ a congruence on $\mathbf{B}$. Then we have the homomorphism $\varphi\times\varphi : \mathbf{A}\times\mathbf{A} \to \mathbf{B}\times\mathbf{B}$ defined by $(\varphi \times \varphi)(a, b) = (\varphi(a), \varphi(b))$, for all $(a, b) \in A \times A$. Because θ is a subalgebra of $\mathbf{B} \times \mathbf{B}$, then $(\varphi \times \varphi)^{-1}\theta$ is a subalgebra of $\mathbf{A} \times \mathbf{A}$, and it is easy to prove that it is a congruence on $\mathbf{A}$.

A *variety of V-congruence filters* is a sequence $\Gamma = (\Gamma_n)_{n\geqslant 1}$ such that

C.1) Γ_n is a filter of V-congruences on $Con\mathbf{F}_nV$ contained in Con_nV;

C.2) If $\varphi : \mathbf{F}_mV \to \mathbf{F}_nV$ is a homomorphism and $\theta \in \Gamma_n$, then $(\varphi \times \varphi)^{-1}\theta \in \Gamma_m$.

To simplify the terminology, we will call a variety of V-congruence filters just a variety of V-congruences, and will represent the lattice of all varieties of V-congruence filters by $\mathcal{VC}(V)$.

Proposition 1. [1] *The correspondences*

$$(\ )^{\ell} : \mathcal{L}^{ps}(V) \to \mathcal{VL}(V) \qquad W \mapsto W^{\ell} = (\{L \subseteq F_nV : \mathbf{F}_nV/L \in W\}_n)_{n\geqslant 1}$$

and

$$(\ )^{a} : \mathcal{VL}(V) \to \mathcal{L}^{ps}(V) \qquad \mathscr{V} \mapsto \mathscr{V}^{a} = V_f\{\mathbf{A}/L \in V : L \in \mathscr{V}_n, n \geqslant 1\}$$

are mutually inverse lattice isomorphisms.

Proposition 2. [1] *The correspondences*

$$(\ )^c : \mathcal{L}^{ps}(V) \to \mathcal{VC}(V) \qquad W \mapsto W^c = (\{\theta \in Con_n V : \mathbf{F}_n V/\theta \in W\}_n)_{n\geqslant 1}$$

and

$$(\ )^a : \mathcal{VC}(V) \to \mathcal{L}^{ps}(V) \qquad \Gamma \mapsto \Gamma^a = V_f\{\mathbf{F}_n V/\theta : \theta \in \Gamma_n, n \geqslant 1\}$$

are mutually inverse lattice isomorphisms.

Where $V_f\{K\}$ denotes the pseudovariety generated by the class of finite algebras K. From the above two isomorphisms we get the next result.

Theorem 1. [1] *The lattices $\mathcal{L}^{ps}(V)$, $\mathcal{VL}(V)$ and $\mathcal{VC}(V)$ are all complete and isomorphic to each other.*

3 Hypersubstitutions and M-solid pseudovarieties

A mapping $\sigma : \{f_i : i \in I\} \to T_\tau(X_\omega)$, which assigns to every n_i-ary operation symbol f_i an n_i-ary term $\sigma(f_i)$, will be called a *hypersubstitution* of type τ. If σ is a hypersubstitution, then we can think of σ as mapping each term of the form $f_i(x_1, \ldots, x_{n_i})$ to the n_i-ary term $\sigma(f_i)$. This means that any hypersubstitution σ induces a unique map $\hat{\sigma}$ on the set $T_\tau(X)$ of all terms of type τ over X, as follows:

(1) $\hat{\sigma}[x] := x$, for all $x \in X$;

(2) $\hat{\sigma}[f_i(t_1, \ldots, t_{n_i})] = \sigma(f_i)(\hat{\sigma}[t_1], \ldots, \hat{\sigma}[t_{n_i}])$, for the term $f_i(t_1, \ldots, t_{n_i})$.

We denote by $Hyp(\tau)$ the set of all hypersubstitutions of type τ. We can define a composition operation $\circ_h$ on hypersubstitutions by $\sigma_1 \circ_h \sigma_2 := \hat{\sigma}_1 \circ \sigma_2$, for $\sigma_1, \sigma_2 \in Hyp(\tau)$. Considering the identity hypersubstitution σ_{id}, the set of all hypersubstitutions form a monoid $\mathbf{Hyp}(\tau) = \langle Hyp(\tau); \circ_h, \sigma_{id}\rangle$. In the sequel, let $M \subseteq Hyp(\tau)$ be a submonoid of hypersubstitutions.

Hypersubstitutions can be applied to an equation $t \approx s \in Eq(\tau)$ to produce a new equation $\hat{\sigma}[t] \approx \hat{\sigma}[s]$. From an algebra $\mathbf{A} = \langle A, (f_i)_{i\in I}\rangle$ of type τ and a hypersubstitution $\sigma \in M$ it is possible to construct a new algebra $\sigma[\mathbf{A}] = \langle A; (\sigma(f_i)^{\mathbf{A}})_{i\in I}\rangle$ called the *M-derived algebra* of $\mathbf{A}$ by σ.

It is easy to see that, if a map φ is a homomorphism $\varphi : \mathbf{A} \to \mathbf{B}$, then it is also a homomorphism $\varphi : \sigma[\mathbf{A}] \to \sigma[\mathbf{B}]$, for all $\sigma \in Hyp(\tau)$. Related to congruences, if $\theta \in Con\mathbf{A}$ is a congruence on $\mathbf{A}$, then θ is also a congruence on $\sigma[\mathbf{A}]$, and $\sigma[\mathbf{A}/\theta] = \sigma[\mathbf{A}]/\theta$.

Definition 1. *A pseudovariety V of algebras of type τ is called* M-solid *if it is closed under M-derived algebras.*

Clearly, the pseudovarieties $Alg_f(\tau)$ of all finite algebras and $I(\tau)$ of all trivial algebras are, respectively, the greatest and smallest M-solid pseudovarieties of type τ. We will represent the set of all M-solid pseudovarieties of type τ by $\mathcal{S}^M_{ps}(\tau)$.

Theorem 2. [4] *For every monoid $M \subseteq Hyp(\tau)$ of hypersubstitutions the set $\mathcal{S}_{ps}^{M}(\tau)$ is a complete sublattice of the lattice $\mathcal{L}_{ps}(\tau)$ of all pseudovarieties of type τ.*

At this point, using Eilenberg-type correspondences we just know that the complete sublattice $\mathcal{S}_{M}^{ps}(V) := \mathcal{S}_{ps}^{M} \cap \mathcal{L}_{ps}(V)$ of all M-solid subpseudovarieties of V corresponds to a complete sublattice of $\mathcal{VL}(V)$ and another complete sublattice of $\mathcal{VC}(V)$. In the next section we give a characterization of this sublattices.

4 M-solid varieties of languages and congruences

We start with the definition of a special kind of hypersubstitution introduced by Płonka in [10].

Definition 2. *Let K be a class of algebras of type τ. A hypersubstitution $\sigma \in Hyp(\tau)$ is called* K-proper *if for all $t \approx s \in Id(K)$ we have $\hat{\sigma}[t] \approx \hat{\sigma}[s] \in Id(K)$. Let $P(K)$ be the set of all K-proper hypersubstitutions.*

We have that, for any class K of algebras, $\mathbf{P}(K) = \langle P(K); \circ_h, \sigma_{id} \rangle$ is a submonoid of $\mathbf{Hyp}(\tau)$. When K is an equational pseudovariety, $P(K)$ is the greatest submonoid of hypersubstitutions such that K is $P(K)$-solid.

The notion of semi-weak homomorphism of an algebra is introduced by Kolibiar in [9].

Definition 3. *Let $\mathbf{A}$ and $\mathbf{B}$ be algebras of type τ. A mapping $h : A \to B$ is called a* semi-weak homomorphism *if there exists a hypersubstitution $\sigma \in Hyp(\tau)$ such that h is a homomorphism of $\mathbf{A}$ into $\sigma[\mathbf{B}]$. In this case, we write $h : \mathbf{A} \xrightarrow{sw} \mathbf{B}$. A semi-weak homomorphism $h : \mathbf{A} \xrightarrow{sw} \mathbf{A}$ is called a* semi-weak endomorphism *of $\mathbf{A}$. We say that $h : \mathbf{A} \to \sigma[\mathbf{B}]$ is an M-semi-weak homomorphism, if $\sigma \in M$.*

Clearly, the extension of a hypersubstitution $\hat{\sigma}$ is a semi-weak endomorphism of $\mathbf{T}_\tau(X)$, and also, any usual homomorphism is a semi-weak homomorphism. We have the following fact.

Proposition 3. *Let $\sigma \in P(V)$ be a V-proper hypersubstitution. Then $\hat{\sigma} : \mathbf{F}_n V \to \mathbf{F}_n V$ is a semi-weak endomorphism, for any $n \geqslant 1$.*

Proof. Let $\mathbf{T}_\tau(X_n)$ be the algebra of n-ary terms of type τ. We have the homomorphism $\hat{\sigma} : \mathbf{T}_\tau(X_n) \to \sigma[\mathbf{T}_\tau(X_n)]$. The V-free algebra $\mathbf{F}_n V$ is given by $\mathbf{F}_n V \cong \mathbf{T}_\tau(X_n)/\theta_V(X_n)$, where the congruence $\theta_V(X_n) := \{(t,s) \in T_\tau(X_n) \times T_\tau(X_n) : t \approx s \in Id(V)\}$ is given by all the equations over X_n satisfied by V. We have that $\theta_V(X_n)$ is also a congruence on $\sigma[\mathbf{T}_\tau(X_n)]$ and that $\sigma[\mathbf{T}_\tau(X_n)]/\theta_V(X_n) = \sigma[\mathbf{T}_\tau(X_n)/\theta_V(X_n)]$. Since the equations satisfied by V are preserved by V-proper hypersubstitutions, we have the homomorphism $\hat{\sigma} : \mathbf{F}_n V \to \sigma[\mathbf{F}_n V]$. Hence, $\hat{\sigma} : \mathbf{F}_n V \xrightarrow{sw} \mathbf{F}_n V$ is an $P(V)$-semi-weak endomorphism. □

Now we give the definition of an M-solid variety of languages.

Definition 4. *Let $\mathscr{V} = (\mathscr{V}_n)_{n\geqslant 1}$ be a variety of V-languages. The variety $\mathscr{V}$ is called an* M-solid variety of V-languages *if for all $n, m \geqslant 1$, and all M-semi-weak homomorphism $h : \mathbf{F}_m V \to \mathbf{F}_n V$ and all $L \in \mathscr{V}_n$ we have $h^{-1}(L) \in \mathscr{V}_m$.*

It is easy to see that the trivial variety of V-languages $LtriV = (\{\emptyset, F_n V\})_n$ is M-solid, for any M. Using the Eilenberg-type correspondences between pseudovarieties and varieties of languages we have the following result.

Proposition 4. *Let W be an M-solid subpseudovariety of V. Then, W^ℓ is an M-solid variety of V-languages.*

Proof. Let W be an M-solid subpseudovariety of V. From Proposition 1 we have that W^ℓ is a variety of V-languages. For $n, m \geqslant 1$, let $h : \mathbf{F}_m V \to \mathbf{F}_n V$ be an M-semi-weak homomorphism, and $L \in W_n^\ell$. We want to show that $h^{-1}(L) \in W_m^\ell$. Since $L \in W_n^\ell$, then $\mathbf{F}_n V/L \in W$. So, L is W-recognizable, and there exists a finite algebra $\mathbf{A} \in W$, a homomorphism $\varphi : \mathbf{F}_n V \to \mathbf{A}$, and a set $K \subseteq A$ such that $\varphi^{-1}(K) = L$. We have the homomorphism $h : \mathbf{F}_m V \to \sigma[\mathbf{F}_n V]$ and as well the homomorphism $\varphi : \sigma[\mathbf{F}_n V] \to \sigma[\mathbf{A}]$, for a hypersubstitution $\sigma \in M$. Then, $h^{-1}(L) = h^{-1}(\varphi^{-1}(K)) = (\varphi \circ h)^{-1}(K)$. So, $h^{-1}(L)$ is recognized by $\langle \sigma[\mathbf{A}], \varphi \circ h, K)$. As a consequence $\mathbf{F}_m V/h^{-1}(L) \preceq \sigma[\mathbf{A}]$. Since W is M-solid, we have that $\sigma[\mathbf{A}] \in W$ and then $\mathbf{F}_m V/h^{-1}(L) \in W$. Hence $h^{-1}(L) \in W_m^\ell$. This proves that W^ℓ is an M-solid variety of V-languages. □

From the last proposition we conclude the following results.

Corollary 1. *Let $L \in Rec_n V$ be a V-recognizable language and $h : \mathbf{F}_m V \to \mathbf{F}_n V$ an M-semi-weak homomorphism. Then $h^{-1}(L)$ is recognized by $\sigma[\mathbf{A}]$ for some hypersubstitution $\sigma \in M$.*

Corollary 2. *Let $\mathscr{V}$ be a variety of V-languages. If the pseudovariety $\mathscr{V}^a$ is M-solid then $\mathscr{V}$ is an M-solid variety of V-languages.*

As claimed in [12] every subdirectly irreducible algebra is a syntactic algebra. Hence, each finite algebra of V can be represented as a subdirect product of a finite number of syntactic algebras of some subsets, that are V-recognized by the algebra. So, as remarked in [11], for any variety $\mathscr{V}$ of V-languages and any finite algebra $\mathbf{A}$, if for all n, all V-languages in $T_\tau(X_n)$ recognized by $\mathbf{A}$ are in $\mathscr{V}_n$, then $\mathbf{A} \in \mathscr{V}^a$.

Proposition 5. *Let V be an equational pseudovariety. If $\mathscr{V}$ is an M-solid variety of V-languages, then the pseudovariety $\mathscr{V}^a$ is $M \bigcap P(V)$-solid.*

Proof. Let $\mathbf{A} \in \mathscr{V}^a$ be any finite algebra and $\sigma \in M \bigcap P(V)$ a hypersubstitution. We will prove that $\sigma[\mathbf{A}] \in \mathscr{V}^a$. From the previous remark, we have only to prove that all V-languages recognized by $\sigma[\mathbf{A}]$ are in $\mathscr{V}$. For any $n \geqslant 1$, let $L \in Rec_n V$ be a V-language recognized by $\sigma[\mathbf{A}]$. Hence, there exists a homomorphism $\varphi : \mathbf{F}_n V \to \sigma[\mathbf{A}]$ and a subset $K \subseteq A$ such that $\varphi^{-1}(K) = L$. Since $\sigma \in P(V)$, then $\sigma[\mathbf{A}] \in V$ and $\hat{\sigma} : \mathbf{F}_n V \to \mathbf{F}_n V$ is an M-semi-weak endomorphism of $\mathbf{F}_n V$.

Let $\psi : \mathbf{F}_nV \to \mathbf{A}$ be the unique homomorphism such that $\psi \circ \hat{\sigma} = \varphi$. Thus, $L = \varphi^{-1}(K) = (\psi \circ \hat{\sigma})^{-1}(K) =$
$= \hat{\sigma}^{-1}(\psi^{-1}(K))$. Since $\mathbf{A}$ recognizes the language $L' = \psi^{-1}(K)$, then $\mathbf{F}_nV/L' \preceq \mathbf{A}$, and as consequence $L' \in \mathscr{V}_n$. Since $\mathscr{V}$ is M-solid, this implies that $L = \hat{\sigma}^{-1}(L') \in \mathscr{V}_n$. We have proved that all V-languages recognized by $\sigma[\mathbf{A}]$ are in $\mathscr{V}$, and so $\sigma[\mathbf{A}] \in \mathscr{V}^a$. Hence, $\mathscr{V}^a$ is an $M \bigcap P(V)$-solid pseudovariety. □

The condition imposed on V, that V be an equational pseudovariety, isn't very restrictive because in both cases, string languages and tree languages, we are dealing with equational pseudovarieties.

The analogous definition of M-solidity for varieties of congruences is presented.

Definition 5. *Let $\Gamma = (\Gamma_n)_{n\geqslant 1}$ be a variety of V-congruences. The variety Γ is said to be M-solid if for all $n, m \geqslant 1$, M-semi-weak homomorphism $h : \mathbf{F}_mV \to \mathbf{F}_nV$ and $\theta \in \Gamma_n$, then $(h \times h)^{-1}\theta \in \Gamma_m$.*

Clearly, the trivial variety of congruences $CtriV = (\{\nabla_{F_nV}\})_n$ is M-solid.

We need these technical results to prove the next proposition.

Lemma 1. [2] *Let $\varphi : \mathbf{A} \to \mathbf{B}$ be a homomorphism, $L \subseteq B$ a subset and $\theta \in Con\mathbf{A}$ a congruence.*

i) If $\sim_L$ has a finite number of classes, then there is only a finite number of subsets that may be obtained from L by cancellation;

ii) $(\varphi \times \varphi)^{-1} \sim_L = \bigcap_{L'}\{\sim_{\varphi^{-1}L'}: L' \text{is obtained from } L \text{ by cancellation}\}$;

iii) $\theta = \bigcap_L\{\sim_L: L \text{ is a class of } \theta\}$.

Lemma 2. [12] *Let $\mathscr{V}$ be a variety of V-languages. For any $n \geqslant 1$ and $L \in \mathscr{V}_n$ all the $\sim_L$-classes are in $\mathscr{V}_n$.*

To connect varieties of languages and varieties of congruences we have this interesting result.

Proposition 6. *Let $\mathscr{V}$ be a variety of V-languages. Then, $\mathscr{V}$ is M-solid iff $\mathscr{V}^c$ is M-solid.*

Proof. ($\Rightarrow$) For $n, m \geqslant 1$, let $h : \mathbf{F}_mV \xrightarrow{sw} \mathbf{F}_nV$ be an M-semi-weak homomorphism and $\theta \in \mathscr{V}_n^c$. Using Lemma 1 it follows that

$$(h \times h)^{-1}\theta = \bigcap_L \bigcap_p \{\sim_{h^{-1}(p^{-1}(L))}: p \in Pol_1(\mathbf{F}_nV)\ L \text{ is a } \theta\text{-class}\}$$

is a finitary intersection. From Lemma 2 we conclude that all classes of θ are in $\mathscr{V}_n$. Since $\mathscr{V}$ is a variety of V-languages we have $p^{-1}(L) \in \mathscr{V}_n$ for all $p \in Pol_1(\mathbf{F}_nV)$. Moreover, $\mathscr{V}$ is M-solid so $h^{-1}(p^{-1}(L)) \in \mathscr{V}_m$ for all $p \in Pol_1(\mathbf{F}_nV)$ and for all θ-classes L. Hence, $(h \times h)^{-1}\theta \in \mathscr{V}_m^c$, which proves that $\mathscr{V}^c$ is M-solid.

($\Leftarrow$) Suppose $\mathscr{V}$ is a variety of V-languages such that $\mathscr{V}^c$ is an M-solid variety of V-congruences. For $n, m \geqslant 1$, let $L \in \mathscr{V}_n$ and $h : \mathbf{F}_m V \to \mathbf{F}_n V$ be an M-semi-weak homomorphism. We have the homomorphism $h : \mathbf{F}_m V \to \sigma[\mathbf{F}_n V]$ for some hypersubstitution $\sigma \in M$. It is easy to prove that $(h \times h)^{-1} \sim_L \subseteq \sim_{h^{-1}(L)}$. Since, $\sim_L \in \mathscr{V}_n^c$ and $\mathscr{V}^c$ is M-solid, then $(h \times h)^{-1} \sim_L \in \mathscr{V}_m^c$. Now, because $\mathscr{V}_m^c$ is a filter of congruences, we conclude that $\sim_{h^{-1}(L)} \in \mathscr{V}_m^c$. Hence, $h^{-1}(L) \in \mathscr{V}_m^{c\ell} = \mathscr{V}_m$. So we have proved that $\mathscr{V}$ is M-solid. □

Corollary 3. *A variety Γ of V-congruence filters is M-solid iff Γ^ℓ is an M-solid variety of V-languages.*

When V is the pseudovariety of all finite algebras of type τ, the V-languages are precisely the recognizable tree languages of type τ. In this case we can prove the next result.

Theorem 3. *Let $V = Alg_f(\tau)$ be the pseudovariety of all finite algebras of type τ. If W is any pseudovariety of type τ the following conditions are equivalent:*

i) W is an M-solid pseudovariety;

ii) W^ℓ is an M-solid variety of tree languages;

iii) W^c is an M-solid variety of congruences.

Proof. This proof is straightforward. We only need to use the previous results and the fact that $Hyp(\tau)$ is the set of all $Alg_f(\tau)$-proper hypersubstitutions. □

The next result shows how to obtain complete sublattices of the lattice of all varieties of tree languages $\mathcal{VL}(\tau)$ and the lattice of all varieties of congruences $\mathcal{VC}(\tau)$. Let $\mathcal{VL}_M(\tau)$ denote the set of all M-solid varieties of tree languages and by $\mathcal{VC}_M(\tau)$ we will represent the set of all M-solid varieties of congruences.

Corollary 4. *The sets $\mathcal{VL}_M(\tau)$ and $\mathcal{VC}_M(\tau)$ are complete sublattices of the complete lattices $\mathcal{VL}(\tau)$ and $\mathcal{VC}(\tau)$, respectively, and both are isomorphic to the complete lattice $\mathcal{S}_M^{ps}(\tau)$ of all M-solid pseudovarieties of type τ.*

We know that any semi-weak homomorphism $h : \mathbf{T}_\tau(X_m) \to \mathbf{T}_\tau(X_n)$ is a homomorphism $h : \mathbf{T}_\tau(X_m) \to \sigma[\mathbf{T}_\tau(X_n)]$ for some hypersubstitution $\sigma \in Hyp(\tau)$. Hence, there exists an endomorphism $\varphi : \mathbf{T}_\tau(X_n) \to \mathbf{T}_\tau(X_n)$ such that $h = \varphi \circ \hat{\sigma}$. Using this fact, in the case of tree languages, M-solid varieties of tree languages and M-solid varieties of congruences have an alternative and more simple characterization given by hypersubstitutions.

Proposition 7. *A variety $\mathscr{V}$ of tree languages is M-solid iff it is closed under inverse hypersubstitution with respect to M, i.e. iff for all $n \geqslant 1$, and any hypersubstitution $\sigma \in M$ and any $L \in \mathscr{V}_n$, we have $\hat{\sigma}^{-1}(L) \in \mathscr{V}_n$.*

A similar result holds for varieties of congruences.

Proposition 8. *Let $\Gamma \in \mathcal{VC}(\tau)$ be a variety of congruences. Then, Γ is M-solid iff for all $n \geqslant 1$, all $\theta \in \Gamma_n$ and all $\sigma \in M$, we have $(\hat{\sigma} \times \hat{\sigma})^{-1}\theta \in \Gamma_n$.*

An important notion in the theory of tree language is the tree homomorphism [13]. Semi-weak homomorphisms and the extensions of hypersubstitutions are particular cases of tree homomorphisms, and each monoid of hypersubstitutions can define a special set of tree homomorphisms. So, the M-solid varieties of tree languages can be characterized using tree homomorphisms.

5 Examples

Now we will see two well known cases of tree languages that are M-solid, for some monoids $M \subseteq Hyp(\tau)$ of hypersubstitutions.

5.1 Nilpotent algebras

An algebra $\mathbf{A}$ of type τ is called *nilpotent*, if there exists an absorbing element a_0 and an integer $k \geq 1$ such that $t^{\mathbf{A}}(a_1, \ldots, a_n) = a_0$ for all terms t in n variables with $hg(t) \geq k$ and for all n-tuples of elements $(a_1, \ldots, a_n)$, where hg is the usual height function on terms. The smallest k for which this holds is called the degree of nilpotency of $\mathbf{A}$. Let $Nil(\tau)$ be the class of all nilpotent algebras of type τ.

For each $n \geqslant 1$, let $\mathcal{N}il_n(\tau)$ consist of all finite and all cofinite tree languages in $T_\tau(X_n)$. For string languages it is well-know that finite and cofinite language are recognizable, and that they form a variety of languages. In the case of tree languages we have the following result.

Proposition 9. [12] *The sequence $\mathcal{N}il(\tau) = (\mathcal{N}il_n(\tau))_{n\geqslant 1}$ is a variety of tree languages and $\mathcal{N}il(\tau)^a = Nil(\tau)$.*

A hypersubstitution σ is called regular if for each f_i, all the variables $x_1, \ldots, x_{n_i}$ occur in $\sigma(f_i)$. The set $Reg(\tau)$ of all regular hypersubstitutions is a submonoid of $Hyp(\tau)$, and applying induction over a term t we prove that if σ is a regular hypersubstitution then $hg(\hat{\sigma}[t]) \geqslant hg(t)$.

Using this lemma we can easily prove the next result about the pseudovariety $Nil(\tau)$.

Proposition 10. *The pseudovariety $Nil(\tau)$ is $Reg(\tau)$-solid.*

Proof. By proposition 9 we know already that $Nil(\tau)$ is a pseudovariety. So, we only have to show that it is closed under $Reg(\tau)$-derived algebras. Let $\mathbf{A} \in Nil(\tau)$ be a nilpotent algebra of degree k and $\sigma \in Reg(\tau)$ a regular hypersubstitution. For any $t \in T_\tau(X_\omega)$ and by induction on the height of t it is easy to prove that $t^{\sigma[\mathbf{A}]} = \hat{\sigma}[t]^{\mathbf{A}}$. Using this fact, for any $t \in T_\tau(X_\omega)$ of $hg(t) \geqslant k$ then $hg(\hat{\sigma}[t]) \geqslant hg(t) \geqslant k$, and we have $t^{\sigma[\mathbf{A}]} = \hat{\sigma}[t]^{\mathbf{A}} = a_0$. Hence, $\sigma[\mathbf{A}]$ is a nilpotent algebra of degree k, and thus $\sigma[\mathbf{A}] \in Nil(\tau)$. □

By proposition 3 we can conclude the following result.

Corollary 5. *The variety of tree languages $\mathcal{N}il(\tau)$ is $Reg(\tau)$-solid.*

5.2 Definite tree languages

A string language L is called k-definite, if a word of length greater then k is in L iff it suffix of length k is in L. The extension to tree languages of this notion is made using roots. For any term t, $root(t) = t$ if t is a variable, and $root(t) = f_i$ if $t = f_i(t_1, \ldots, t_{n_i})$, for some $i \in I$.

The k-root $R_k(t)$ of a term $t \in T_\tau(X_\omega)$ is defined as follows:

i) $R_0(t) = \varepsilon$, where ε is a special symbol which represents an empty root and $R_1(t) = root(t)$;

ii) let $k \geqslant 2$, if $hg(t) < k$ and $t = f_i(t_1, \ldots, t_{n_i})$ for some $i \in I$, then $R_k(t) = t$. If $hg(t) \geqslant k$ then $R_k(t) = f_i(R_{k-1}(t_1), \ldots, R_{k-1}(t_{n_i}))$.

Let $k \geqslant 0$ and $L \subseteq T_\tau(X_n)$. The language L is called k-definite, if for all $t, s \in T_\tau(X_n)$ such that $R_k(t) = R_k(s)$, $t \in L$ iff $s \in L$. For each $n \geqslant 0$, we define the relations $\sim_{k,n}$ in $T_\tau(X_n)$ as follows:

$$t \sim_{k,n} s \quad \text{iff } R_k(t) = R_k(s),$$

for all $t, s \in T_\tau(X_n)$. We will simply denote $t \sim_k s$, if $R_k(t) = R_k(s)$.

Let $\mathcal{D}^k(\tau) = (\mathcal{D}_n^k(\tau))_{n \geqslant 1}$ be a sequence, where $\mathcal{D}_n^k(\tau)$ is the set of all k-definite tree languages of $T_\tau(X_n)$. The sequence of all definite tree languages is $\mathcal{D}(\tau) = (\mathcal{D}_n(\tau))_{n \geqslant 1}$, where $\mathcal{D}_n(\tau) = \bigcup\{\mathcal{D}_{k,n}(\tau) : k \geqslant 0\}$ is the set of all definite tree languages of $T_\tau(X_n)$. Clearly, we can conclude the inclusions

$$\mathcal{D}^0(\tau) \subseteq \mathcal{D}^1(\tau) \subseteq \cdots \subseteq \mathcal{D}^k(\tau) \subseteq \cdots \subseteq \mathcal{D}(\tau).$$

Thus, we have the following result.

Proposition 11. [12] *For all $n \geqslant 1$ and $k \geqslant 0$ the relation $\sim_{k,n}$ is a congruence on the algebra $\mathbf{T}_\tau(X_n)$, and it has a finite number of classes.*

Let $\Gamma_{k,n} = [\sim_{k,n})$ be the principal filter generated by $\sim_{k,n}$ on the lattice $FCon\mathbf{T}_\tau(X_n)$ of all finite index congruence relations of $\mathbf{T}_\tau(X_n)$.

Proposition 12. [12] *For each $k \geqslant 0$ the sequence $\Gamma_k = (\Gamma_{k,n})_{n \geqslant 1}$ is a variety of congruences. Moreover, $\Gamma_k^\ell = \mathcal{D}^k(\tau)$.*

From this last proposition we conclude that for any $k \geqslant 0$ the sequences $\mathcal{D}^k(\tau)$ and $\mathcal{D}(\tau)$ are varieties of tree languages.

A hypersubstitution σ is called a *pre-hypersubstitution* if for every $i \in I$ the term $\sigma(f_i)$ is not a variable. The set of all pre-hypersubstitutions $Pre(\tau)$ is a submonoid of $Hyp(\tau)$. The next result shows an important behavior of pre-hypersubstitutions related to preservations of k-roots.

Lemma 3. *Let $t, s \in T_\tau(X)$ and $\sigma \in Pre(\tau)$. If $t \sim_k s$ then $\hat{\sigma}[t] \sim_k \hat{\sigma}[s]$, for all $k \geqslant 0$.*

Proof. This proof is made by induction on k. The first case $k = 0$ is obvious. Let $k \geqslant 1$ and suppose it is true for $k-1$. Let $\sigma \in Pre(\tau)$ and $t, s \in T_\tau(X_n)$, such that $t \sim_k s$. If t or s are variables then they must be the same variable and clearly $\hat{\sigma}[t] \sim_k \hat{\sigma}[s]$. If not, we must have $t = f_i(t_1, \ldots, t_{n_i})$ and $s = f_i(s_1, \ldots, s_{n_i})$ for some $i \in I$ such that $t_1 \sim_{k-1} s_1, \ldots, t_{n_i} \sim_{k-1} s_{n_i}$. Hence, we have $\hat{\sigma}[t] = \sigma(f_i)(\hat{\sigma}[t_1], \ldots, \hat{\sigma}[t_{n_i}])$ and $\hat{\sigma}[s] = \sigma(f_i)(\hat{\sigma}[s_1], \ldots, \hat{\sigma}[s_{n_i}])$. By induction hypothesis $\hat{\sigma}[t_1] \sim_{k-1} \hat{\sigma}[s_1], \ldots, \hat{\sigma}[t_{n_i}] \sim_{k-1} \hat{\sigma}[s_{n_i}]$. Because $\sigma(f_i)$ is not a variable, and $\sim_k$ is a congruence we can conclude that $\hat{\sigma}[t] \sim_k \hat{\sigma}[s]$. □

Proposition 13. *For each $k \geqslant 0$ the sequence $\Gamma_k = (\Gamma_{k,n})_{n \geqslant 1}$ is a pre-solid variety of congruences.*

Proof. Let $k \geqslant 0$. By proposition 8, we need to prove that for every pre-hypersubstitution $\sigma \in Pre(\tau)$ and $\theta \in \Gamma_{k,n}$, we have $(\hat{\sigma} \times \hat{\sigma})^{-1}\theta \in \Gamma_{k,n}$, for each $n \geqslant 1$. So, it is sufficient to show that $\sim_{k,n} \subseteq (\sigma \times \sigma)^{-1}\theta$. Let $t, s \in T_\tau(X_n)$ such that $t \sim_{k,n} s$. By the previous lemma we have $\hat{\sigma}[t] \sim_{k,n} \hat{\sigma}[s]$, and so $(t, s) \in (\varphi \times \varphi)^{-1} \sim_{k,n} \subseteq (\varphi \times \varphi)^{-1}\theta$. Hence, $\Gamma_k = (\Gamma_{k,n})_{n \geqslant 1}$ is a pre-solid variety of congruence filters. □

Now, we are able to state the next result.

Corollary 6. *For any $k \geqslant 0$, the varieties of tree languages $\mathcal{D}^k(\tau)$ and $\mathcal{D}(\tau)$ are pre-solid varieties of tree languages.*

This Corollary follows from Corollary 11.13 of [7].

6 Conclusion

The approaches of Ésik [7] and Steinby [13] generalize the Eilenberg-type correspondence to tree languages; they are not restricted to a fixed algebraic type. In particular, Ésik uses algebraic theories, and Steinby adds some constructions to the usual Universal Algebra. It is easy to see that the $*$-varieties and $+$-varieties of Ésik correspond to solid and pre-solid varieties of languages, respectively, as presented here. Steinby's general varieties correspond to hypersubstitutions which map the n-ary operational symbols to primitive terms $f_i(x_1, \ldots, x_n)$ (with a change in the algebraic type). We believe that hypersubstitutions and M-solid pseudovarieties are an adequate generalization of the aforementioned approaches and that they are suitable for characterizing varieties of tree languages. In order to see this, one needs to work with hypersubstitutions between different algebraic types which form a small category. Then it is easy to generalize all the other notions presented here. But, to go even further and generalize these results to positive and many-sorted varieties [11], it is necessary to work within the framework of Institutions [8].

7 Acknowledgments

The author wishes to express his special gratitude to Margarita Ramalho whose support and supervision made this work possible. Thanks also to S. Wismath and M. Branco by reading an earlier version of the paper, and by their valuable suggestions and remarks. This work was developed within the Project POCTI-ISFL-1-143 "Fundamental and Applied Algebra" of Centro de Algebra da Universidade de Lisboa, financed by FCT and FEDER. The author is supported by FCT and EU FEDER PhD fellowship SFRH/BD/22698/2005.

References

[1] Almeida, J. On pseudovarieties, varieties of languages, filters of congruences, pseudoidentities and related topics, *Algebra Universalis*, 27:333–350, 1990.

[2] Almeida, J. Finite Semigroups and Universal Algebra, *Word Scientific*, 1994.

[3] Baltazar, P. Variedades M-sólidas de linguagens. *Master thesis*, 2005.

[4] Denecke, K. and Pibaljommee, B. M-solid pseudovarieties and galois connections, *Advances in Algebra. Proceedings of the ICM satellite Conference in Algebra and related topics, World Scientific, Hong Kong*, pages 325–342, 2003.

[5] Denecke, K. and Reichel, M. Monoids of hypersubstitutions and M-solid varieties, *Contributions to General Algebra*, Verlag Hölder-Pichler-Tempsky, Wien, 9:117-126, 1995.

[6] Eilenberg, S. Automata, Languages and Machines, *Vol. B Pure and Applied Mathematics, Vol. 59*, Academic Press, New York - London, 1976.

[7] Ésik, Z. A variety theorem for trees and theories, *Publ. Math.*, 54(1–2): 711–762, 1999.

[8] Goguen, J. and Burstall, R. Institutions: Abstract model theory for specification and programming, *Journal of the ACM*, 39(1): 95–146, 1992.

[9] Kolibiar, M. Weak homomorphism in some classes of algebras, *Studia Sci. Math. Hung.*, 19:413–420, 1984.

[10] Płonka, J. Proper and inner hypersubstitutions of varieties, *Proceedings of the International Conference Summer School on General Algebra and Ordered Sets*, Olomouc, pages 106–116, 1994.

[11] Saheli, S. Varieties of Tree Languages, *PhD Dissertation*, Department of Mathematics, University of Turku, TUCS Dissertations 64, 2005.

[12] Steinby, M. A theory of tree language varieties, *in: Nivat M. & Podelski A.(ed.) Tree automata and languages*, Elsevier-Amsterdam, pages 57-81, 1992.

[13] Steinby, M. General varieties of tree languages, *Theor. Comput. Sci.*, 205(1–2): 1–43, 1998.

[14] Thérien, D. Recognizable languages and congruences, *Semigroup Forum*, 23:371–373, 1981.

Received 27th October 2006

Acta Cybernetica 18 (2008) 733–776.

Effect Preservation in Transaction Processing in Rule Triggering Systems*†

Mira Balaban‡ and Steffen Jurk§

Abstract

Rules provide an expressive means for implementing database behavior: They cope with changes and their ramifications. Rules are commonly used for integrity enforcement, i.e., for repairing database actions in a way that integrity constraints are kept. Yet, Rule Triggering Systems fall short in enforcing *effect preservation*, i.e., guaranteeing that repairing events do not undo each other, and in particular, do not undo the original triggering event.

A method for enforcement of effect preservation on updates in general rule triggering systems is suggested. The method derives transactions from rules, and then splits the work between compile time and run time. At compile time, a data structure is constructed, that analyzes the execution sequences of a transaction and computes minimal conditions for effect preservation. The transaction code is augmented with instructions that navigate along the data structure and test the computed minimal conditions.

This method produces *minimal effect preserving transactions*, and under certain conditions, provides meaningful improvement over the quadratic overhead of pure run time procedures. For transactions without loops, the run time overhead is linear in the size of the transaction, and for general transactions, the run time overhead depends linearly on the length of the execution sequence and the number of loop repetitions. The method is currently being implemented within a traditional database system.

Keywords: rule triggering systems, effect preservation, minimal conditions, consistency, static analysis, transaction processing

*This work was supported in part by the Paul Ivanir Center for Robotics and Production Management at Ben-Gurion University of the Negev. Contact: POB 653, Beer Sheva 84105, ISRAEL, Phone: +972-8-6472222, FAX: +972-8-6477527.

†This research was supported by the DFG, Berlin-Brandenburg Graduate School in Distributed Information Systems (DFG grant no. GRK 316). Contact: POB 101433, 03013 Cottbus, Germany, Phone: +49-355-692711, FAX: +49-355-692766

‡Ben-Gurion Univerity, Beer-Sheva, Israel, E-mail: mira@cs.bgu.ac.il

§Brandenburg University of Technology, Cottbus, Germany, E-mail: sj@informatik.tu-cottbus.de

1 Introduction

Rules provide an expressive means for implementing database behavior: They cope with changes and their ramifications. Rule mechanisms are used in almost every commercial database system, using features such as `CREATE TRIGGER` or `CREATE RULE`. Rules are commonly used for *Integrity enforcement*, i.e., for *repairing* database actions in a way that integrity constraints are kept ([37, 16, 38]). Yet, *Rule Triggering Systems* (*RTSs*), also termed *Active Databases*, fall short in enforcing *effect preservation*, i.e., guaranteeing that repairing events do not undo each other, and in particular, do not undo the original triggering event.

A natural expectation in database maintenance is that a fact that was successfully added is retrievable, as long as it was not intentionally removed. Reliability in question answering, and faithfulness to the intended semantics of operations, require no contradictory operations. Such behavior is achievable if rule application that can cause contradictory updates is avoided. Active databases do not meet this expectation since it is possible that a rule application undoes the actions of previous rule applications in a single repair transaction. Moreover, in a distributed active database a user might not be aware of rules that trigger contradicting actions, since the rules might reside in independently developed sites.

Example 1. Consider a database with a table T_1 with two attributes A, B, a table T_2 with an attribute C, and two integrity constraints: an inclusion constraint ($A \subseteq C$) and an exclusion constraint ($B \cap C = \emptyset$)[1]. The inclusion constraint is enforced by the rules:

$$R_1: \quad \text{ON} \quad insert(T_1, (x, \_)) : \text{IF } (x) \notin T_2 \text{ THEN } insert(T_2, (x))$$
$$R_2: \quad \text{ON} \quad delete(T_2, (x)) : \text{IF } (x) \in T_1.A \text{ THEN } delete(T_1, (x, \_))$$

The exclusion constraint is enforced with the rules:

$$R_3: \quad \text{ON} \quad insert(T_1, (\_, x)) : \text{IF } (x) \in T_2 \text{ THEN } delete(T_2, (x))$$
$$R_4: \quad \text{ON} \quad insert(T_2, (x)) : \text{IF } (x) \in T_1.B \text{ THEN } delete(T_1, (\_, x))$$

An insertion update $insert(T_1, (a, b))$ triggers rules R_1 and R_3, in order to preserve the integrity constraints. The value a is inserted into T_2 (if not exists) to preserve the inclusion constraint, which in turn causes a deletion of tuples $(\_, a)$ from T_1 in order to preserve the exclusion constraint. Analogously the value b is removed from T_2 (if exists) to preserve the exclusion constraint which causes deletion of tuples $(b, \_)$ from T_1. If the event-repair policy is that a rule is fired immediately AFTER its event (triggering update) is executed, and the repairing rules R_1 and R_3 are triggered in that ordering, the order of rule application for this insertion event is R_1, R_4, R_3, R_2. A database design tool would result the following repair update:

[1] these constraints are simplified versions of possibly more natural constraints like $f(A) \subseteq C$ and $g(B) \cap C = \emptyset$, where f and g are some functions.

```
INSERT INTO T1 VALUES (a,b);
IF NOT EXISTS (SELECT * FROM T2 WHERE C=a) THEN
       INSERT INTO T2 VALUES (a);
       DELETE FROM T1 WHERE B=a;
END IF;
IF EXISTS (SELECT * FROM T2 WHERE C=b) THEN
       DELETE FROM T2 VALUES (b);
       DELETE FROM T1 WHERE A=b;
END IF;
```

If $a = b$, then the repairing update might undo its own actions, since tuples inserted to the tables might be deleted:

Rule	Triggering Primitive Update	T_1	T_2
–	–	(a,a)	$\emptyset$
R_1	$insert(T_1,(a,a))$	(a,a)	(a)
R_4	$insert(T_2,(a))$	$\emptyset$	(a)
R_3	$insert(T_1,(a,a))$	$\emptyset$	$\emptyset$
R_2	$delete(T_2,(a))$	$\emptyset$	$\emptyset$

If the rule application ordering is R_3, R_1, R_4, the result is that the inserted tuple is deleted from T_1, while a new tuple is still inserted to T_2 as a repair for the deleted insertion to T_1:

Rule	Triggering Primitive Update	T_1	T_2
–	–	(a,a)	$\emptyset$
R_3	$insert(T_1,(a,a))$	(a,a)	$\emptyset$
R_1	$insert(T_1,(a,a))$	(a,a)	(a)
R_4	$insert(T_2,(a))$	$\emptyset$	(a)

This example demonstrates the problem of *effect preservation*. A seemingly successful insertion update ends up in a state where the insertion is not performed. Moreover, although the insertion actually failed, its repairing updates have taken place. The problem is caused by allowing contradictory updates, i.e., updates that undo the expected effects of each other, within the context of a successful repairing transaction. Rather, the insertion $insert(T_1,(a,a))$ must fail (be rejected) since there is no way to achieve consistency together with effect preservation.

The problem of *effect preservation* goes back to the early days of *Planning* in Artificial Intelligence, when planners tried to cope with the problem of *interacting goals*, where one action undoes something accomplished by another (e.g., Strips [15], Noah [31]). In the field of databases, [32, 35] provide a general framework for consistency enforcement under an effect preservation constraint. Static composition of refactoring ([20, 19]) also needs to cope with possibly contradicting effects of successive refactorings.

In the field of active databases the problem of effect violation occurs when a rule fires other rules that perform updates that contradict an update of a previous rule. Nevertheless, automated generation and static analysis of active database rules

[37, 6, 12] do neither handle, nor provide a solution for that problem. Updates triggered within the scope of a single repair can contradict each other.

In this paper we suggest a combined, compile time - run time, method for enforcing effect preservation on updates. The method is applied in two steps:

1. **Static derivation of transactions from rules**: For each primitive (atomic) update, e.g., insertion or deletion of a tuple, obtain a transaction, based on a given rule application policy.

2. **Effect Preservation Transformation**: Enforce effect preservation using "minimal" modifications.

Our method assumes that primitive (atomic) updates are associated with intended effects (postconditions). At compile time, we construct a data structure that analyzes all execution sequences of an update and computes minimal conditions necessary for effect preservation. The update code is augmented with instructions that navigate along the data structure and test the computed minimal conditions. This method produces *minimal effect preserving transactions*, and under certain conditions, provides meaningful improvement over the quadratic overhead of pure run time procedures. For transactions without loops, the run time overhead is linear in the size of the transaction, and for general transactions, the run time overhead depends linearly on the length of the execution sequence and the number of loop repetitions (while for pure run time methods it is necessarily quadratic in the length of the execution sequence).

Our method is domain independent, but in this paper we demonstrate it on the relational domain with element insertion and deletion operations, alone. The method is currently being implemented within a traditional database system.

Section 2 introduces a small imperative update language *While* that we use in the rest of this paper, and describes the algorithm for static derivation of transactions from rules. Section 3 introduces the major notions of effect preservation. In Section 4 algorithms for effect preservation of *While* updates are described and proved to enforce only *minimal effect preservation*, i.e., do not cause unnecessary failures. The algorithms are introduced gradually, first for *While* updates without loops, and then for general *While* updates. Section 5 describes related work, and section 6 concludes the paper. Proofs are postponed to the Appendix.

2 Static Rule Repair

Rule repair in active databases is performed at run time. The database tool is usually equipped with an *event-repair policy* (e.g., an AFTER policy), while the order of firing the applicable rules, henceforth *rule-ordering policy*, is determined at run time. The event-repair policy determines when an event that is applied by a rule is repaired. For example, using a DELAYED event-repair policy in Example 1, and SEQUENTIAL rule-ordering policy, the order of rule application for the $insert(T_1, (a, b))$ insertion event is R_1, R_3, R_4, R_2. The rule-ordering policy is

responsible for sequencing the set of rules that should repair for an event. For example, using a DELAYED event-repair policy and a REVERSED SEQUENTIAL rule-ordering policy in Example 1, the order of rule application for the $insert(T_1, (a, b))$ insertion event is R_3, R_1, R_4.

If the rule-ordering policy is determined statically, the overall task of repairing for an event can be determined statically, by associating every database event with a transaction. The advantage is that an update transaction that is constructed at compile time can be also optimized at compile time, thereby saving some run time load.

Subsection 2.1 describes the small theoretical imperative update language *While* [26], that is used in the rest of the paper. Subsection 2.2 introduces a compile time algorithm for derivation of updates from rules. The rest of the paper deals with enforcing effect preservation on statically derived updates.

2.1 The Repair Language *While* – An Imperative Language of Updates (Transactions)

The *While* language [26] is a small theoretical imperative language that includes the three major control structures in sequential imperative languages: sequencing, conditionals and loops. We adapt it for database update usage by adding a *fail* atomic statement that stands for *rollback*, and by having state variables that stand for relations. In the experimental evaluation the language is replaced by a real database maintenance language. Likewise, all the examples in the paper deal with a relational database and the atomic updates *insert* and *delete* of a tuple to a relation. However, our method applies to any atomic updates for which effects can be provided (see subsection 3.1).

Syntax: Updates are built over a finite set of typed *state variables* For example, in a relational database with relations $R_1, \ldots, R_n$, the state variables are $\{R_1, \ldots, R_n\}$, and any assignment of concrete relations to the relation variables results a *database state*. The symbols of the *While* language include, besides the state variables, *parameter variables, local variables*, and *constant symbols* like numerals, arithmetic operations and comparisons, boolean constants and boolean connectives, and set operations (*language constants*). We use the letters r, s, t for state variables, $u, v, w, \ldots$ for parameter or local variables, and $e, f, g, \ldots$ for expressions.

The primitive update statements of *While* are *skip*, *fail*, and *well-typed assignments*. The *skip* update is a no-op update statement: Its execution does not affect the state of the update. The *fail* update denotes a failure of the update: Updates following *fail* are not executed. In transactional databases without choice, *fail* corresponds to the *rollback* operation, which undoes the failed update by restoring the old state. Assignment updates have the form $x := e$, where x is a variable and e is an expression.

Compound updates (transactions) in the *While* language are formed by three constructors: *sequence, condition*, and *while*. If P is a condition, and S, S_1, S_2

are updates, then "$S_1; S_2$" is the sequential composition of S_1 and S_2, "`if` P `then` S_1 `else` S_2" is a P conditioned update, and "`while` P `do` S" is a P conditioned S loop, with S as the body of the loop. The statement "`if` P `then` S" is an abbreviation for "`if` P `then` S `else` *skip*". The *empty update* ϵ is the neutral (unit) element of update sequencing. That is, for every update S, $\epsilon; S \equiv S; \epsilon \equiv S$, where $\equiv$ stands for "is the same as" or "can be replaced by" (ϵ is needed for the definition of the terminal configurations below). A compound update S with parameter variables $\vec{x}$ is sometimes denoted $S(\vec{x})$.

Semantics: A *state* is a well-typed *value assignment* to all variables. A *database state* is a restriction of a state to the state variables. An non-ground expression (that includes variables) can be evaluated only with respect to a state. The value of an expression e in a state s is denoted e^s. A state s that satisfies a condition P is denoted as $s \models P$. Variable substitution on a state s is denoted $s[x \mapsto e^s]$, which is a state that differs from s only in the value of x which is e^s.

The semantics of *While* updates is defined operationally, using the *structural operational semantics* described in [26]. This semantical approach emphasizes individual steps of the execution, that are captured by a *transition relation* denoted $\Rightarrow$ between update configurations. An *update configuration* (*configuration* for short) is a pair $\langle S, s\rangle$ of an update S and a state s. A configuration $\langle \epsilon, s\rangle$ is called a *terminal configuration* and abbreviated as s. All non-terminal configurations are *intermediate*. A configuration $\langle fail, s\rangle$ is called a *failing configuration*. A configuration $\langle A; S, s\rangle$ where A is an assignment update and S is any update (possibly the empty update), is called an *assignment configuration*.

The *transition relation* between configurations $\langle S, s\rangle \Rightarrow \gamma$, expresses the first step in the execution of S from state s. If the execution of S from s is not completed by the transition, γ is an intermediate configuration $\langle S', s'\rangle$, where S' is the remaining computation to be executed from state s'. If the execution of S from s is completed by the transition, γ is a terminal configuration $\langle \epsilon, s'\rangle \equiv s'$. A non-terminal configuration γ is a *dead-end* if there is no γ' such that $\gamma \Rightarrow \gamma'$. Failing configurations are dead-end configurations. The transition relation $\Rightarrow$ is defined by the axioms and rules in Figure 1.

An *execution sequence* (denoted as Ψ, Φ or $seq(S, s)$) of a statement S starting in state s, is either a finite or an infinite sequence of configurations. In a finite execution sequence, $seq(S, s) = \gamma_0, \ldots, \gamma_k$, where $\gamma_0 = \langle S, s\rangle$, $\gamma_i \Rightarrow \gamma_{i+1}$ $(0 \leq i < k)$ and γ_k is either a terminal or a dead-end configuration. In an infinite execution sequence, $seq(S, s) = \gamma_0, \ldots,$ where $\gamma_0 = \langle S, s\rangle$, and $\gamma_i \Rightarrow \gamma_{i+1}$, $0 \leq i$. A finite sequence is *successful* if it ends in a terminal configuration, and *failing* if it ends in a dead-end configuration. The states of the first and last (if exists) configurations of an execution sequence Ψ are denoted $start(\Psi)$ and $end(\Psi)$, respectively, and the i-th configuration $(0 \leq i)$ is denoted Ψ_i.

Extended *While*: *While* is extended with calls to external procedures that do not affect the state of a *While* computation. Such external procedures can operate

$[ass]$ $\quad \langle x := e, s\rangle \Rightarrow s[x \mapsto e^s]$

$[skip]$ $\quad \langle skip, s\rangle \Rightarrow s$

$[fail]$ $\quad \langle fail, s\rangle$

$[comp]$ $\quad$ **if** $\langle S_1, s\rangle \Rightarrow \langle S_1', s'\rangle$ **then** $\langle S_1; S_2, s\rangle \Rightarrow \langle S_1'; S_2, s'\rangle$
if S_1' **is** ϵ **then the conclusion is** $\langle S_1; S_2, s\rangle \Rightarrow \langle S_2, s'\rangle$

$[if^T]$ $\quad \langle \texttt{if } P \texttt{ then } S_1 \texttt{ else } S_2, s\rangle \Rightarrow \langle S_1, s\rangle$ **if** $s \models P$

$[if^F]$ $\quad \langle \texttt{if } P \texttt{ then } S_1 \texttt{ else } S_2, s\rangle \Rightarrow \langle S_2, s\rangle$ **if** $s \not\models P$

$[while]$ $\quad \langle \text{while } P \text{ do } S, s\rangle \Rightarrow$
$\langle \texttt{if } P \texttt{ then } (S; \text{ while } P \text{ do } S) \texttt{ else skip}, s\rangle$

Figure 1: Axioms and rules defining the semantics of the transition relation $\Rightarrow$

on elements of an external environment, e.g., print or draw. Formally, the extended *While* includes an additional primitive update *proc*, where *proc* is a procedure that can be applied in the environment where *While* is run. The semantics of *proc* is like that of *skip*:

$$\langle proc, s\rangle \Rightarrow s$$

The transformations introduced in Section 4 map a *While* update without external calls into a *While* update with external calls to procedures that read and evaluate data from a data structure built as part of the transformation.

2.2 Static Derivation of Transactions from Rules

Event-Condition-Action (*ECA*) rules consist of *event*, *condition*, and *action* (also termed *body*), with the intuitive semantics that an execution of the event update implies an execution of the action update, provided that the condition holds. For simplicity, we embed the *condition* part into the *action* part, since an ECA rule $\langle E, C\rangle \rightarrow A$ can be captured by the rule $E \rightarrow \texttt{if } C \texttt{ then } A$.

Compile time rule application requires careful management of parameter and local variables. Prior to rule application, all such variables must be consistently renamed by fresh variables. Then, the expression in the triggering event must unify with the expression in the rule event, and the resulting substitution applied to the rule body. For example, if the update under construction includes the primitive update $insert(r, x+3)$, and the event in an applicable rule is $insert(r, x)$, then the variable x must be renamed in the rule by a new name, say y, yielding the event $insert(r, y)$. The two events $insert(r, x+3)$ and $insert(r, y)$ should be unified,

yielding the matching substitution $y \mapsto x + 3$, which should be applied to the rule body, replacing all occurrences of y by $x + 3$.

Compile time rule processing builds for each primitive event a complex update, that exhausts all necessary rule applications. It requires *static event-repair policy, rule-ordering policy,* and *static termination policy.* A termination policy is necessary since rule triggering can be cyclic. Much research have been devoted to termination analysis ([36, 5]). In this work we take the approach of full propagation, that is controlled by a static termination condition.

Algorithm 1 below, performs static derivation of transactions from rules. It is designed for the AFTER event-repair semantics, which characterizes commercial databases. The AFTER semantics is implemented by sequencing immediately after every occurrence of an assignment to a state variable event, the bodies of all rules that are applicable to that event. Rule application handles variable renaming, rule event matching and substitution application.

Algorithm 1. [*DTA* – **Derive TransAction**]
input: *a primitive update U, a set of rules $\mathcal{R}$, a rule-ordering policy RO, and a termination condition TC.*
output: *A While program S.*
method:

1. $S := U$

2. *While there is an unmarked assignment E in S do:*

   a) *Mark E.*
   // The marking is used to avoid multiple replacements of the same primitive update.

   b) *For every rule in $\mathcal{R}$, consistently rename all parameter and local variables, by fresh variables.*

   c) *Let $E_1 \rightarrow B_1, \ldots, E_k \rightarrow B_k$ be all rules whose event can be unified with E, and are such ordered by RO. Let ϕ_i be the matching substitution for E and E_i, i.e., $E\phi_i = E_i\phi_i$, $1 \leq i \leq k$.*

   d) *If $k \neq 0$ and $\neg TC$: Replace E in S by $E; B_1\phi_1; \ldots; B_k\phi_k$*

   e) *If $k \neq 0$ and TC: Replace E in S by $E; fail$*

3. *Return S*

The following example presents the rules from Example 1 as event-action rules in the *While* language, and the update constructed by the *DTA* algorithm, for the triggering event $insert(T_1, (x, y))$, using a sequential rule-ordering policy.

Example 2 (Static transaction derivation with the DTA algorithm.).

$$
\begin{aligned}
R_1 &: \mathit{insert}(T_1,(x,y)) \rightarrow \texttt{if } (x) \notin T_2 \texttt{ then } \mathit{insert}(T_2,(x));\\
R_2 &: \mathit{delete}(T_2,(x)) \rightarrow \texttt{while } \sigma_{A=x}(T_1) \neq \emptyset \texttt{ do}\\
&\qquad\qquad \mathit{delete}(T_1,\mathrm{first}(\sigma_{A=x}(T_1)));\\
R_3 &: \mathit{insert}(T_1,(x,y)) \rightarrow \texttt{if } (y) \in T_2 \texttt{ then } \mathit{delete}(T_2,(y));\\
R_4 &: \mathit{insert}(T_2,(x)) \rightarrow \texttt{while } \sigma_{B=x}(T_1) \neq \emptyset \texttt{ do}\\
&\qquad\qquad \mathit{delete}(T_1,\mathrm{first}(\sigma_{B=x}(T_1)));
\end{aligned}
$$

The *While* program derived for the update $\mathit{insert}(T_1,(x,y))$:

$$
\begin{aligned}
&\mathit{insert}(T_1,(x,y));\\
&\texttt{if } (x) \notin T_2 \texttt{ then}\\
&\quad \mathit{insert}(T_2,(x));\\
&\quad \texttt{while } \sigma_{B=x}(T_1) \neq \emptyset \texttt{ do}\\
&\quad\quad \mathit{delete}(T_1,\mathrm{first}(\sigma_{B=x}(T_1)));\\
&\texttt{if } (y) \in T_2 \texttt{ then}\\
&\quad \mathit{delete}(T_2,(y));\\
&\quad \texttt{while } \sigma_{A=y}(T_1) \neq \emptyset \texttt{ do}\\
&\quad\quad \mathit{delete}(T_1,\mathrm{first}(\sigma_{A=y}(T_1)));
\end{aligned}
$$

Assignment Preprocessing: Prior to the transaction derivation a preprocessing of rule actions is needed. Our effect characterization methods require that every assignment to a state variable $r := e(r,x_1,\ldots,x_n)$ is rewritten such that the variables $x_1,\ldots,x_n$ are not assigned after the assignment is executed. The rationale behind this requirement is that each such assignment has effects which are a set of constraints, expressed by the variables in the assignment. If the variables $x_1,\ldots,x_n$ are reassigned after the assignment is executed, then the effects of the assignment must be reassigned as well. In order to avoid this hard procedure, we rewrite the assignment, using new variables, as follows:
The assignment $r := e(r,x_1,\ldots,x_n)$, is replaced by the sequence update

$$x'_1 := x_1;\ \ldots,\ x'_n := x_n;\ r := e(r,x'_1,\ldots,x'_n);$$

where $x'_1,\ldots,x'_n$ are new variables. The new sequence command has the same semantics as the original assignment, when restricted to the original update variables.

3 Effect Preservation – Goals and Associated Terminology

In this section we introduce the main concepts of *effect preservation* and analyze their basic properties. We deal with three main issues: (1) Define what are effects

and where do they come from; (2) Define when an update is effect preserving; (3) Define criteria for effect preserving transformations. The section ends with the definition of a *minimal effect preserving transformation.*

3.1 Characterization of Effects

The effects of a primitive update are constraints (postconditions) that characterize the intended impact of the update:

Definition 1 (Effects of Primitive Updates). *The effects of a primitive update U is a collection of First Order Logic formulae effects(U) that always hold following the update. That is, for every state s, if $\langle U, s\rangle \Rightarrow s'$, then $s' \models effects(U)$*[2].

Effects are used as a criterion for acceptance or rejection of execution sequences. Therefore, if they are too restrictive, they lead to needless rejections, while if they are too permissive, they lead to unwanted acceptance that possibly violates some data integrity constraints. For example, taking $\{false\}$ as the effects of an update causes constant rejection, while taking $\{true\}$ causes constant acceptance.

For the primitive *skip* the effects are $\{\}$ since $\langle skip, s\rangle \Rightarrow s$. For the primitive *fail* the effects can be anything since, $\langle fail, s\rangle$ is a dead-end configuration. For assignments to regular variables the effects are also $\{\}$ since such updates have no impact on database states. For assignments to state variables, the effects are domain specific and developer provided. In the relational domain with the operations of element insertion and deletion we assume the following effects:

- $effects(r := insert(r, e))$ is $\{e \in r\}$ and
- $effects(r := delete(r, e))$ is $\{e \notin r\}$.

Clearly, different effects can be considered. For example, the effects of element insertion and deletion can be the dynamic constraints: If $\{e \notin r\}$ ($\{e \in r\}$) before the assignment, then $\{e \in r\}$ ($\{e \notin r\}$) after the assignment, respectively. Yet, these effects are still questionable, and in the current work we do not handle dynamic effects.

3.2 Definition of Effect Preservation

Consider the update:

$$\begin{array}{l} insert(r, e_1); \\ \text{if } P \text{ then } insert(r, e_2) \text{ else } skip; \\ delete(r, e_3) \end{array}$$

Starting from any state, there are two possible execution sequences, based on whether P holds after the first insertion. Effect preservation requires that both sequences preserve the $\{e_1 \in r\}$ condition, but only one should preserve the $\{e_2 \in r\}$

[2] For a set of formulae Φ and a state s, $s \models \Phi$ holds iff for every formula ϕ in Φ, $s \models \phi$.

condition. Therefore, effect preservation requires distinction between execution sequences. That is, the definition of effect preservation should be execution sequence sensitive. This observation leads us to the following definition:

Definition 2 (The effect preservation property of updates). *An update S is Effect Preserving if all of its execution sequences are effect preserving.*

Now we need to define effect preservation for execution sequences. For that purpose, we need first to assign to a configuration within an execution sequence, the effects of previous assignments along the sequence, so to maintain the history of the sequence.

Definition 3 (Effects of Configurations within Execution Sequences). *Let $\Psi = \gamma_0, \gamma_1, \ldots$ be an execution sequence. The effects of the i-th configuration in Ψ, denoted $effects_i(\Psi)$, are:*
$effects_0(\Psi) = \{\}$, and for $i \geq 0$:

$$effects_{i+1}(\Psi) = \begin{cases} effects_i(\Psi) \cup effects(A) & \text{if } \gamma_i \text{ is an assignment configuration } \langle A; S, s_i \rangle \text{ to a state variable} \\ effects_i(\Psi) & \text{otherwise} \end{cases}$$

For a successful (finite length k) execution sequence, the effects are those of its last configuration: $effects(\Psi) = effects_k(\Psi)$. For a failing (finite) execution sequence the effects are $\{false\}$. Note that the effects set of an assignment configuration does not include the effects of its assignment update.

Property 1. *For a successful execution sequence Ψ, the size of $effects(\Psi)$ is proportional to the length of Ψ.*

Example 3 (Effects-of-configurations). The effects for the execution sequence

$$\Psi = \langle r := insert(r, x); r := delete(r, y),\ s \rangle \Rightarrow \langle r := delete(r, y),\ s' \rangle \Rightarrow \langle \epsilon,\ s'' \rangle$$

are: $effects_0(\Psi) = \{\}$, $effects_1(\Psi) = \{x \in r\}$, $effects_2(\Psi) = \{x \in r, y \notin r\} = effects(\Psi)$.

Note: Due to the rewriting preprocessing of assignments to state variables (see Subsection 2.2), the effects of configurations indeed maintain the intended history of effects. Consider an assignment $A = r := e(r, y_1, \ldots, y_n)$ to a state variable r. $effects(A)$ imposes a constraint on r in terms of the values of $y_1, \ldots, y_n$ in the state that follows the update execution. If the variables $y_1, \ldots, y_n$ are later on assigned, then $effects(A)$ should be modified to reflect this change. Since the assignment preprocessing guarantees that these variables are not assigned along the sequence, $effects(A)$ can be taken as the final effect of the assignment.

The *effect preservation* property is concerned with maintaining the effects of primitive updates along a sequence:

Definition 4 (The Effect Preservation Property of Execution Sequences). *An execution sequence* $\Psi = \gamma_0, \gamma_1, \ldots$, *where* $\gamma_i = \langle S_i, s_i \rangle$ $i \geq 0$ *is* Effect Preserving (EP) *if it is either failing or* $s_i \models \mathit{effects}_i(\Psi)$ *for all* $i \geq 0$.

Example 4 (Effect preservation).

1. Consider the execution sequence from Example 3. The sequence is effect preserving for $s = [r \mapsto \{3,4,5\}, x \mapsto 3, y \mapsto 4]$ but is not effect preserving for $s = [r \mapsto \{3,4,5\}, x \mapsto 3, y \mapsto 3]$.

   Therefore, the update $r := \mathit{insert}(r,x); r := \mathit{delete}(r,y)$ is not effect preserving.

2. Consider the execution sequence

   $$\langle r := \mathit{insert}(r,x); r := \mathit{insert}(r,y),\ s\rangle \Rightarrow \langle r := \mathit{insert}(r,y),\ s'\rangle \Rightarrow \langle \epsilon,\ s''\rangle$$

   and states s, s', s'' where $s' = s[r \mapsto \mathit{insert}(r,x)^s]$ and $s'' = s'[r \mapsto \mathit{insert}(r,y)^{s'}]$. The sequence is effect preserving for every state s.

   Therefore, the update $r := \mathit{insert}(r,x); r := \mathit{insert}(r,y)$ is effect preserving.

Note that a failing execution sequence is effect preserving, and an infinite execution sequence can be effect preserving as well.

Proposition 1. *An execution sequence* $\Psi = \langle S_0, s_0\rangle, \langle S_1, s_1\rangle, \ldots$ *in which* $s_{i+1} \models \mathit{effects}_i(\Psi)$ *for all* $i \geq 0$, *is effect preserving.*

3.3 Criteria for Effect Preserving Transformations

The first question that we need to answer when coming to define effect preserving transformation concerns the kind of transformations we wish to produce. One option is *semantics based code transformation*, e.g., the replacement of $r := \mathit{insert}(r,e); r := \mathit{delete}(r,e)$ by *skip*[3]. However, in this case, we need to set criteria for the desired transformation. For example, Schewe and Thalheim in [32, 35], require that all post conditions of the input update are preserved by the transformed code. But in that case, even a simple update such as $r := \mathit{insert}(r,e); r := \mathit{delete}(r,e)$ cannot be replaced by *skip* since, for example, in the state $s = [r \mapsto \{3,4,5\}, e \mapsto 3]$, the input update has the postcondition $\{e \notin r\}$ which is not satisfied by *skip*.

Therefore, we adopt a more modest approach where non preservation of effects causes rejection. We term it "repair by rejection". We still need to characterize desirable repairs since otherwise, the simplest transformation can repair every non effect preserving update by replacing it with *fail*. We set two criteria for good effect preserving transformations:

[3]This is the approach taken in the Generalized Consistent Specialization theory of [32, 35]. This approach is discussed in Section 5.

1. Minimize rejections inserted for enforcing effect preservation.
2. Minimize run time overhead, i.e., minimize tests overhead.

The first criterion is handled by introducing a *minimal restriction relation* between updates. The restriction relation reflects the desired criterion for minimizing the enforced rejections. An effect preserving transformation is required to produce an update which is a minimal restriction of the input update. The second criterion is handled by replacing update effects by *delta-conditions*, which are minimal conditions that guard against effect violation. A desired effect preserving transformation should produce a minimal restriction using delta-conditions.

3.3.1 Update Restriction – Minimizing Enforced Rejections

We already noted that effect preservation should be execution sequence sensitive. That is, it should take into consideration the different effects of different execution sequences. Therefore, the restriction relation on updates is defined on the basis of a restriction relation on execution sequences.

Restriction Relations on Execution Sequences

We introduce three increasingly tighter relations, termed *restriction, EP-restriction* and *minimal-EP-restriction.* The *restriction* relation on execution sequences relates sequences that are either both infinite or terminate in the same state, or one is failing. The *EP-restriction* relation further requires that the restricting sequence is EP. The *minimal-EP-restriction* relation is an EP-restriction with minimal rejections (no needless rejections).

Definition 5 (Restriction relations between execution sequences)**.**

1. *An execution sequence* Ψ' *is a* restriction *of an execution sequence* Ψ*, denoted* $\Psi' \leq \Psi$*, if*
   a) $start(\Psi') = start(\Psi)$,
   b) *If* Ψ' *is infinite then* Ψ *is also infinite,*
   c) *If* Ψ' *is successful with* $end(\Psi') = s'$ *then also* Ψ *is successful with* $end(\Psi) = s'$.

   That is, Ψ' *might be failing while* Ψ *is successful and terminates properly, or is infinite.*
2. *An execution sequence* Ψ' *is an* EP-restriction *of an execution sequence* Ψ*, denoted* $\Psi' \leq_{EP} \Psi$*, if* $\Psi' \leq \Psi$*, and* Ψ' *is EP.*
3. *An execution sequence* Ψ' *is a* minimal-EP-restriction *of an execution sequence* Ψ*, denoted* $\Psi' \leq_{min\_EP} \Psi$*, if* $\Psi' \leq_{EP} \Psi$*, and if* Ψ *is EP then* $\Psi \leq_{EP} \Psi'$ *(i.e.,* $\Psi' \equiv \Psi$*).*

The relations $\leq, \leq_{EP}, \leq_{min\_EP}$ have *proper* versions, denoted $<, <_{EP}, <_{min\_EP}$, respectively. $\Psi' < \Psi$ means that Ψ' is failing while Ψ is successful or infinite. Also, Ψ and Ψ' are *equivalent*, denoted $\Psi' \equiv \Psi$, if $\Psi' \leq \Psi$, and $\Psi \leq \Psi'$. $\Psi' \equiv \Psi$ means that Ψ and Ψ' are both either failing or successful or infinite, and if they are successful they end in the same state. Ψ and Ψ' are *EP-equivalent*, denoted $\Psi' \equiv_{EP} \Psi$, if $\Psi' \leq_{EP} \Psi$, and $\Psi \leq_{EP} \Psi'$. That is, $\Psi' \equiv_{EP} \Psi$ if $\Psi \equiv \Psi'$ are both effect preserving. Note that a failing execution sequence is a restriction of any other sequence.

Proposition 2. *The relations $\equiv$ and $\equiv_{EP}$ are equivalence relations on execution sequences with a common start state, and the $\leq$, $\leq_{EP}$ and $\leq_{min\_EP}$ relations are partial orders with respect to $\equiv$.*

Proof. Immediate. □

Example 5 (Minimal-EP-restriction of execution sequences).

1. Consider the execution sequence Ψ from Example 3. Assume: $s = [r \mapsto \{3,4,5\}, x \mapsto 3, y \mapsto 4]$. The following execution sequence is a minimal-EP-restriction of Ψ:

   $\langle r := insert(r,x);\ \ r := delete(r,y);\ if\ x \notin r\ then\ fail\ else\ skip,\ s\rangle \Rightarrow$
   $\langle r := delete(r,y);\ if\ x \notin r\ then\ fail\ else\ skip,\ s'\rangle \Rightarrow$
   $\langle if\ x \notin r\ then\ fail\ else\ skip,\ s''\rangle \Rightarrow$
   $\langle skip,\ s''\rangle \Rightarrow$
   $\langle \epsilon,\ s''\rangle$

2. Consider the execution sequence

   $\Psi = \langle r := insert(r,e);\ r := delete(r,e),\ s\rangle \Rightarrow \langle r := delete(r,e),\ s'\rangle \Rightarrow \langle \epsilon,\ s\rangle.$

   Assume: $s = [r \mapsto \{3,4,5\}, e \mapsto 4]$. Two minimal-EP-restrictions of ψ are:

   $$\langle skip, s\rangle \Rightarrow \langle \epsilon, s\rangle \ \leq_{min\_EP}\ \Psi \text{ and}$$
   $$\langle fail, s\rangle \ \leq_{min\_EP}\ \Psi.$$

   However, while $\langle fail, s\rangle <_{EP} \langle skip, s\rangle \Rightarrow \langle \epsilon, s\rangle$,
   we have $\langle fail, s\rangle \not\leq_{min\_EP} \langle skip, s\rangle \Rightarrow \langle \epsilon, s\rangle$.

Restriction Relations on Updates

Definition 6 (Restriction relations between updates).

1. *An update U' is a* restriction *of an update U, denoted $U' \leq U$, if for all states s, $seq(U', s) \leq seq(U, s)$.*

2. *An update U' is an* EP-restriction *of an update U, denoted $U' \leq_{EP} U$, if for all states s, $seq(U', s) \leq_{EP} seq(U, s)$.*

3. *An update U' is a* minimal-EP-restriction *of an update U, denoted $U' \leq_{min\_EP} U$, if for all states s, $seq(U', s) \leq_{min\_EP} seq(U, s)$.*

In analogy to the restriction relations of execution sequences, the update restriction relations also induce proper versions and equivalence relations. $U' < U$ means that U' has failures which are successful in U. Updates U and U' are *semantically equivalent* (or just *equivalent*, for short), denoted $U' \equiv U$, if for all states s, $seq(U', s) \equiv seq(U, s)$. Update U and U' are *termination equivalent*, denoted $U' \equiv_t U$, if for all states s, the successful sequences are equivalent. That is, there might be states for which one update fails while the other loops.

Proposition 3. *The semantical equivalence is an equivalence relation on the set of updates, and the update relation restriction $\leq$, $\leq_{EP}$ and $\leq_{min\_EP}$ are partial orders on the set of updates, partitioned by the semantical equivalence relation.*

Example 6 (Minimal-EP-restriction of updates).
Consider the update $S = r := insert(r, e);\ r := delete(r, e)$. Since for the state $s = [r \mapsto \{3, 4, 5\}, e \mapsto 3]$ $seq(skip, s) \not\leq_{min\_EP} seq(S, s)$, we have $skip \not\leq_{min\_EP} S$.

Properties of the update restriction relations:

1. For every update U, $fail \leq U$ and $fail \leq_{EP} U$. That is, the set of updates has a single least element with respect to the relations $\leq$ and $\leq_{EP}$.

2. $U' \leq_{EP} U$, if $U' \leq U$, and U' is EP.

3. The restriction relations on updates are increasingly tighter. That is: If $U' \leq_{EP} U$ then also $U' \leq U$. If $U' \leq_{min\_EP} U$ then also $U' \leq_{EP} U$.

4. If $U' \leq_{min\_EP} U$ and U is EP, then $U \leq_{EP} U'$ (i.e., $U' \equiv U$). However, the converse is not true since it is possible that all execution sequences of U' are minimal-EP restrictions of the corresponding execution sequences of U, while still U is not effect preserving (some sequences of U are not EP).

The restriction relations between updates are used for characterizing the desirable update transformations as *minimal effect preserving*:

Definition 7. *An update transformation Θ is minimal effect preserving if for every update U, $\Theta(U) \leq_{min-EP} U$.*

3.3.2 Delta Conditions – Minimizing Run Time Overhead

Effect preservation can be obtained by inserting a test for past effects following every assignment in the sequence. However, the tests for past effects are costly, since the size of the effects of a configuration $effects_i(\Psi)$ in a sequence Ψ is proportional to i. Therefore, the tests overhead in the worst case is $\mathcal{O}(length(\Psi)^2)$, assuming that testing the effects of primitive updates takes constant time.

The tests overhead can be improved by replacing the tests of past effects by *delta-conditions*, which are minimal formulae that serve as guards against effect violation. Our experience shows that delta-conditions tend to be small and independent of the length of the execution sequence. Therefore, based on this experience, their test overhead for the whole sequence is linear in the length of the execution sequence.

Delta-conditions extract the possible interaction between aggregated past effects and a forthcoming assignment. Consider, for example, the execution sequence Ψ from Example 3, where $s = [r \mapsto \{3,4,5\}, x \mapsto 3, y \mapsto 4]$. The first item in Example 5 presents a minimal-EP-restriction for Ψ. Static analysis of the effects of Ψ reveals that there is a single delta-condition $x \neq y$ to be tested prior to the application of the second assignment in Ψ. Therefore, a more efficient minimal-EP-restriction of Ψ is:

$$\begin{array}{l}\langle r := \mathit{insert}(r,x);\ \ \mathit{if}\ x \neq y\ \mathit{then}\ r := \mathit{delete}(r,y)\ \mathit{else}\ \mathit{fail},\ s\rangle \Rightarrow \\ \langle\ \mathit{if}\ x \neq y\ \mathit{then}\ r := \mathit{delete}(r,y)\ \mathit{else}\ \mathit{fail},\ s'\rangle \Rightarrow \\ \langle r := \mathit{delete}(r,y),\ s'\rangle \Rightarrow \\ \langle \epsilon,\ s''\rangle\end{array}$$

First we define delta-conditions in general, and then specialize to the simple relational domain with element insertion and deletion alone.

Definition 8 (Delta-Conditions of Assignments). *A delta-condition of an assignment update A with respect to a set of formulae P is a minimal collection of First Order Logic formulae $\delta(A,P)$ that guarantees that A does not violate P. That is, for every state s, such that $s \models P$ and $s \models \delta(A,P)$, if $\langle A,s\rangle \Rightarrow s'$, then $s' \models P$.*

A delta-condition can be viewed, alternatively, as a first order formula, by taking the conjunction of the formulae in the set. The empty set corresponds to the formula *true*. Henceforth, the exact view of delta-conditions can be understood from the context: A delta-condition in a test is treated as a formula while a delta-condition in a set operation is treated as a set.

Computation of Delta-Conditions: $\delta(A,P)$ is not necessarily unique. The worst (useless) delta-condition for every assignment A and a formula P is $\{\mathit{false}\}$. If A is $x := e$ then $\{P_{x/e}\}$ is another example of a useless delta-condition. The best delta-condition is the empty set, which indicates that the update A does not interfere with P.

Delta-conditions can be computed in a domain specific manner, by considering the assignments and their effects. For the relational domain with element insertions and deletions, and effects formulae of the form $\{e \in r\}$ and $\{e \notin r\}$, $\delta(A,P)$ is computed by the following rules:

1. If $A = r := \mathit{delete}(r,e_1)$ and P includes $e_2 \in r$, then $\delta(A,P)$ includes the formula $e_1 \neq e_2$.

2. If $A = r := \mathit{insert}(r,e_1)$ and P includes $e_2 \notin r$, then $\delta(A,P)$ includes the formula $e_1 \neq e_2$.

Note that if the assigned state variable in A does not occur in P, then $\delta(A, P)$ is $\{\}$. The delta-conditions computed for the examples below are based on these rules.

Definition 9. [Delta-conditions of Configurations within Execution Sequences]

Let $\Psi = \gamma_0, \gamma_1, \ldots$ be an execution sequence. The delta-conditions of an assignment configuration is the delta-conditions between its effects and the assignment updates. For non assignment configuration the delta-condition is empty. For $i \geq 0$:

$$\delta_i(\Psi) = \begin{cases} \delta(A, effects_i(\Psi)) & \text{if } \gamma_i \text{ is an assignment configuration } \langle A; S, s_i \rangle \text{ to a state variable} \\ \{\} & \text{otherwise} \end{cases}$$

Example 7 (Delta-conditions-of-configurations). Consider the execution sequence Ψ from Example 3. While the effects of its configurations are:
$effects_0(\Psi) = \{\}$, $effects_1(\Psi) = \{x \in r\}$, $effects_2(\Psi) = \{x \in r, y \notin r\}$,
the delta-conditions of its configurations are meaningfully smaller:
$\delta_0(\Psi) = \{\}$, $\delta_1(\Psi) = \{x \neq y\}$, $\delta_2(\Psi) = \{\}$.

Size of delta-Conditions of a Configuration in the relational Domain with Element insertion and Deletion: Consider an assignment configuration $\gamma_i = \langle A; S, s_i \rangle$ in an execution sequence $\Psi = \gamma_0, \gamma_1, \ldots$. The size of $\delta_i(\Psi)$ depends on the number of opposite assignments in $\gamma_0, \gamma_1, \ldots, \gamma_{i-1}$. If $A = r := op(r, e_i)$ where $op = insert$ ($delete$), then the size of $\delta_i(\Psi)$ depends on the number of deletions (insertions) to r, respectively. In the worst case, if all configurations before γ_i are opposite operations on r: $r : -inverse\_op(r, e_j)$, then $\delta_i(\Psi) = \{e_j \neq e_i \mid 0 \leq j < i\}$, which is proportional to the length of the execution sequence. Nevertheless, as already commented above, our experience is that multiple contradictory assignments to the same relation in a single execution sequence are rare. Therefore, based on this experience, we assume that the size of delta-conditions is small (bounded), and their tests take constant time.

4 Algorithms for Enforcing Effect Preservation on Updates

In this section we introduce a minimal effect preserving transformation for *While* updates. That is, an input update S is transformed into an update S' such that $S \leq_{min\_EP} S'$. The transformation combines compile time analysis with run time evaluation of delta-conditions. This approach was selected since we already know that:

- Effect preservation requires run time information, as it is execution sequence sensitive. Otherwise, the minimize rejections criterion is not met.
- Naive run time effect preservation has a worst case overhead of $\mathcal{O}(size(S)^2)$ for an update S (assuming that testing the effects of primitive updates takes constant time), since at every modification all past effects must be checked. The delta-conditions optimization does not apply since their computation depends on the full effects of configurations in the actual execution sequence.

Our method includes three steps:

1. Construct a computation tree (graph) that spans all execution sequences of the update.
2. Annotate the computation tree (graph) with delta-conditions.
3. Augment the update with instructions that navigate along the computation tree (graph) and test the computed delta-conditions. The output is an *extended-While* update: A *While* update augmented with external calls.

The transformation is introduced gradually: First, for a restricted subset of *While* without loops, and then it is extended to any *While* update.

4.1 Minimal Effect Preservation for *While* updates without Loops

The effect preserving transformation EP_1 applies two algorithms: *build_CT* for computation tree construction, and $reviseUpdate_1$ for code transformation. *build_CT* constructs a tree, associates it with an iterator (a pointer), and returns a static encoding of the resulting iterator. $reviseUpdate_1$ interleaves within the input code tree navigation (using the iterator) and tests for delta-conditions that are read from the tree. The EP_1 algorithm returns code in the extended *While* – extended with calls to these external procedures.

Algorithm 2. [EP_1 – **Minimal Effect-Preservation for** *While* **without loops**]

input: *A While update S without loops.*
output: *A pair: $\langle$computation tree for S, minimal-EP restriction of $S\rangle$.*
method:

$EP_1(S) =$ $T := build\_CT(S)$; $S' := reviseUpdate_1(S)$;
return $\langle T,\ S'\rangle$

4.1.1 Computation Tree Construction

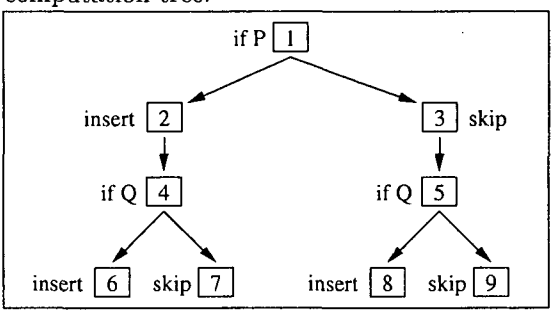
The *computation tree* is a syntax tree whose nodes are associated with delta-conditions. Paths in the tree correspond to execution sequences of the update,

starting from a given state. Nodes correspond to configurations, and are associated with their delta-conditions (note that delta-conditions of configurations are independent from their states).

Example 8 (A computation tree of an update). The update

$S(\text{"A"}, \text{"B"}) = \texttt{if } P \texttt{ then } Car := insert(Car, \text{"A"});$
$\texttt{if } Q \texttt{ then } Car := insert(Car, \text{"B"})$

has the following computation tree:



For this update, the tree captures the four possible execution sequences. Each sequence corresponds to a path of the tree. The effects of the paths are (from left to right): {"A" $\in Car$, "B" $\in Car$}, {"A" $\in Car$}, {"B" $\in Car$}, {}.

The tree construction consists of the actual tree construction, annotation of the tree nodes with effects and delta-conditions, and finally, removal of the effects annotation. The effects annotation is necessary only for the delta-conditions computation. This is important since effects grow linearly with the length of the execution sequence (path in the tree), while delta-conditions tend to be small and independent of path length.

The algorithm uses a node constructor $n = Node(S)$ that returns a node n labeled by S (denoted $label(n)$), and methods $addLeft(n, m)$ ($addRight(n, m)$) that add a left (right) node m to n. The left (right) branch of n are retrieved by $getLeft(n)$ ($getRight(n)$), respectively, and the conjunction of its delta-conditions is retrieved by $getDelta(n)$. The algorithm returns an *iterator* to the tree. The iterator function *node*() returns the currently pointed node, and the functions *right*(), *left*() advance the iterator to the right or left child node of the current node, respectively (if the current node is a leaf node they do nothing). The iterator is necessary for interleaving tree navigation in the transformed update.

Algorithm 3. [*build_CT* – **Build a computation tree for** *While* **without loops**]
input: *A While statement S without loops.*
output: *An iterator for a computation tree for S, initialized to the root node.*
method: $build\_CT(S) = iterator(annotateT(CT(S))$,
where CT and annotateT are the following procedures:

I. CT(S) – *the actual tree construction. CT is defined inductively on the structure of While statements, following the semantics of the transition relation* $\Rightarrow$. *Therefore, the sequence operator* ; *is taken as left associative. That is, in* $S_1; S_2$, S_1 *can be either primitive or an* **if** *statement, since it cannot include the sequence operator.*

1. *For a primitive update* U, $CT(U) = Node(U)$.
2. $CT(\ S_1; S_2\ ) = root$, *where* $root = CT(S_1)$.
   For all leaves l *of root, such that* $label(l) \neq fail$ *do:* $addLeft(l, CT(S_2))$.
3. $CT(\ \texttt{if}\ P\ \texttt{then}\ S_1\ \texttt{else}\ S_2) = root$,
   where $root = Node(\texttt{if}\ P\ \texttt{then}\ S_1\ \texttt{else}\ S_2)$,
   $addLeft(root, CT(S_1))$ *and* $addRight(root, CT(S_2))$.

II. annotateT(T) – *the annotation procedure. It annotates a computation tree* T *built by CT with effects and with delta-conditions with respect to these effects. The effects annotation is only intermediate, and is removed once all the delta conditions are computed.*
$annotateT(T) = remove\_effects(annotateT\_helper(T, \{\}))$, *where,*

$$annotateT\_helper(T, eff) =$$
$$effects(root(T)) = eff,$$
$$\delta(root(T)) = \begin{cases} \delta(label(root(T)), eff) & \text{if } label(root(T)) \text{ is a state variable assignment} \\ \{\} & \text{otherwise} \end{cases}$$
$$eff' = \begin{cases} eff \cup effects(label(root(T))) & \text{if } label(root(T)) \text{ is a state variable assignment} \\ eff & \text{otherwise} \end{cases}$$
$$annotateT\_helper(getLeft(T), eff')$$
$$annotateT\_helper(getRight(T), eff')$$

4.1.2 Code Transformation

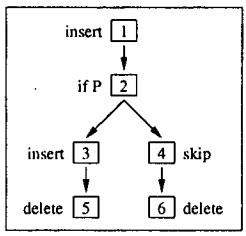
The transformation interleaves within the update code commands for navigating the computation tree and for testing delta-conditions. Tree navigation is done using the iterator operations *node*(), *left*(), *right*(). The statically computed delta-conditions are read from the tree and checked at run time. The transformation is inductively defined on the structure of *While* statements. A syntactic piece of code is denoted `[..]`, and code concatenation is expressed by `[..]` · `[..]`. Evaluable constructs within the syntactic brackets are understood from context.

Algorithm 4. [$reviseUpdate_1$ – **Code transformation for** *While* **without loops**]
input: *A While update S without loops.*
output: *An extended While update which is a minimal-EP restriction of S, when combined with the computation tree of S .*
method:

1. *S is a primitive update:*
   a) *S is an assignment to a state variable:*
      Replace S by [if getDelta(node()) then S; left() else fail];
   b) *Otherwise, S is a regular assignment or skip or fail:*
      Replace S by: [S; left()]
2. $S = [S_1; S_2\ ]$: *Replace S by* $reviseUpdate_1(S_1) \cdot reviseUpdate_1(S_1)$
3. $S =$*[if P then* S_1 *else* S_2 *]: Replace S by*
   [if P then left();] $\cdot\ reviseUpdate_1(S_1)\cdot$
   [else right();] $\cdot\ reviseUpdate_1(S_2)$

Example 9. (The EP_1 Transformation of a non effect preserving update into an effect preserving one.) Consider the following artifical, but comprehensive, update $S(x, y, z)$, where P is and arbitrary condition:

```
S(x,y,z) =
1       r := insert(r,x);
2       if P then r := insert(r,y);
3       r := delete(r,z)
```



Clearly, $r := delete(r, z)$ might violate the effects of both insertions, `insert(r,x)` and `insert(r,y)`, depending on P and the values of x, y, z. The effects and delta-conditions of nodes are as follows:

node	*effects*	*delta-condition*
1	$\{\}$	$\{\}$
2	$\{x \in r\}$	$\{\}$
3	$\{x \in r\}$	$\{\}$
4	$\{x \in r\}$	$\{\}$
5	$\{x \in r, y \in r\}$	$\{x \neq z, y \neq z\}$
6	$\{x \in r\}$	$\{x \neq z\}$

Recall that the effects of a node do not include the effects of the node itself. The delta-conditions of nodes 1, 2, 3 and 4 are empty since there is no previous update whose effects could be violated. The delta-conditions of nodes 5 and 6 are inequalities of the inserted and deleted elements. Applying Algorithm 4 to S returns the following update where the lines are labeled according the original code:

```
S'(x,y,z) =
1           if getDelta(node()) then r := insert(r,x); left() else fail;
2a          if P then
2b            left();
```

```
2c          if getDelta(node()) then r := insert(r, y); left() else fail;
2d        else
2e          right(); skip; left();
3         if getDelta(node()) then r := delete(r, z); left() else fail
```

Where the external calls $getDelta(node())$, $left()$ and $right()$ refer to the computation tree $build\_CT(S)$.

Algorithm 4 can be further statically optimized by removing redundant tests of empty delta-conditions. For this purpose, the $reviseUpdate_1$ procedure should accept also the computation tree as a parameter, and use a tree service $nodes_T(occur(i, U), S)$ that maps the i-th occurrence of a primitive update U in S to the set of tree nodes that correspond to configurations of this occurrence of U. In Example 9, $nodes_T(occur(1, r := insert(r, x)), S) = \{1\}$, and $nodes_T(occur(1, r := insert(r, z)), S) = \{5, 6\}$. Such an optimization simplfies the output update and saves redundant run time tests of delta-conditions. The optimization is obtained by revising the primitive update entry in Algorithm 4, as follows:

If S is the i-th occurrence of a primitive update:

1. S is an assignment to a state variable, and for some $n \in nodes_T(i, S)$, $\delta(n) \neq \emptyset$:
   Replace S by [if $getDelta(node())$ then $S; left()$ else $fail$];

2. Otherwise:
   Replace S by: $[S; left()]$

4.2 Minimal Effect Preservation for *While* updates with Loops

The presence of loops introduces difficulties in detecting past effects since the actual assignments being applied are not made explicit by the syntactic structure, but are determined at run time by loop repetitions. Therefore, we need to strengthen the complie time created structure with information that can capture the dynamics of loops, and extend the update so that it can track the actually visited loops. The nodes of the computation tree, which is extended into a *computation graph*, are annotated with additional delta-conditions with respect to all possibly visited nodes, and the code transformation prepares structures for storing and testing the values of variables at run time.

As before, the effect preserving transformation EP_2 applies two algorithms: $build\_CG$ for computation graph construction, and $reviseUpdate_2$ for code transformation. The EP_2 algorithm returns code in *While*, extended with external calls for navigating the computation graph, testing the delta-conditions, and recording values of variables in loops.

Algorithm 5. [EP_2 – **Minimal Effect-Preservation for** *While*]
input: *A While statement S.*
output: *A pair:* $\langle$*computation graph for S, minimal-EP restriction of S*$\rangle$.
method:

$EP_2(S) =$ $G := build\_CG(S)$; $S' := reviseUpdate_2(S, G)$;
return $\langle G,\ S' \rangle$

4.2.1 Computation Graph Construction

As in Algorithm 3, the graph construction consists of the actual graph construction, annotation of the graph nodes with delta-conditions, and finally, removal of redundant annotation that is necessary only for the delta-condition computation.

Algorithm 6. [*build_CG* – **Build a computation graph for** *While*]
input: *A While statement S.*
output: *An iterator for a computation graph for S, initialized to the root node.*
method: $build\_CG(S) = iterator(annotateG(CG(S)))$,
where CG and annotateG are the following procedures:

I. CG(S) – *the actual graph construction. CG extends the inductive tree construction of procedure CT with a fourth entry for a while update:*

4. $CG($ `while` P `do` $S) = root$, *where* $root = Node($`while` P `do` $S)$,
$addLeft(root, CG($`if` P `then` S `else skip`$))$.
For all leaves l *of* $left(left(root))$, *such that* $label(l) \neq fail$ *do:* $addLeft(l, root)$

The while entry follows the while semantics:

$$\langle \textit{while } P \textit{ do } S, s\rangle \Rightarrow \langle \texttt{if } P \texttt{ then } (S;\ \textit{while } P \textit{ do } S) \texttt{ else skip}, s\rangle$$

Loop repetitions in a while statement are captured by graph cycles. Note that the graphs are finite and have leaf nodes – the leaf in a while statement graph is the right child node of the added `if-then-else` *statement. Therefore, the construction algorithm is well-defined.*

II. annotateG(G) – *the annotation procedure. It annotates a computation graph G built by CG with delta-conditions with respect to its effects. During the annotation phase each node n is associated, in addition to its effects and delta-conditions, with:*

1. $possible(n)$ – *All assignment (to state variable) nodes on cycles through n.*
2. $\delta Possible(n)$ – *Delta-conditions with respect to the effects of the assignments in these nodes. For each node m in* $possible(n)$, *a delta-condition between* $label(n)$ *and* $effects(label(m))$ *is computed.* $\delta Possible(n)$ *is a set of all pairs* $\langle m, \delta_m \rangle$ *where* $m \in possible(n)$ *and* $\delta_m = \delta(label(n), effects(label(m)))$.

Like the effects annotation, the possible(n) annotation, which grows linearly with the length of the execution sequence (path in the graph), is only intermediate, and is removed when the annotation is completed. The only annotations left in the final graph are the $\delta(n)$ and $\delta Possible(n)$ annotations, which tend to be small and independent of path length.

The set possible(n), for a node n, can be computed using any cycle detection algorithm. However, it is not sufficient to consider only simple cycles since cycles through nested loops are not simple (the while node in the cycle must be visited several times). Below we provide separate annotation procedures for the possible and $\delta Possible$ sets. The effects and the delta-conditions annotations are essentially the ones from the tree construction version (Algorithm 3).

$annotateG(G) =$
$remove(annotate\_\delta Possible(annotate\_possible($
$annotate\_effects\_\delta(G))))$

where,

1. *$annotate\_effects\_\delta(G)$: Same as annotateT from the tree construction algorithm. The only difference is the escape from looping. For that purpose, the recursive application of annotate_helper on $getLeft(G)$ should be preceded by a test that the child node was not previously visited. For example, test that $effects(getLeft(G))$ is not defined already.*

2. *Set for all nodes n of the computation graph: $possible(n) = \emptyset$.*
   $annotate\_possible(G) =$

   a) Mark $root(G)$ as visited.

   b) For $n = getLeft(root(G))$ or $getRight(root(G))$:

   - *If n is marked as visited:*
     *If $possible(n) = possible(root(G))$: **Stop.***
     else $possible(n) = possible(n) \cup \{root(G)\}$
   - *$possible(n) = possible(n) \cup possible(root(G))$*

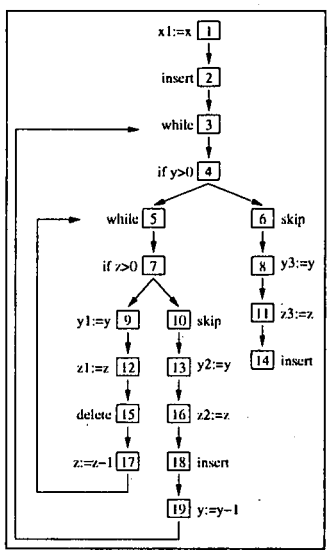
   c) $annotate\_possible(getLeft(root(G)))$
   $annotate\_possible(getRight(root(G)))$

3. *$annotate\_\delta Possible =$*
   For every node n in G, which is an assignment to a state variable node:
   For every node $m \in possible(n)$, which is an assignment to a state variable node:
   Add to $\delta Possible(n)$: $\langle m,\ \delta(label(n), effects(label(m)))\rangle$.

4. *The remove procedure removes the effects and the possible annotations from the graph.*

Example 10 (Annotated computation graph with loops.). A *While* statement and its computation graph.

```
S(x, y, z) =
1     x_1 := x;
2     r := insert(r,x_1);
3     while y > 0 do
4       while z > 0 do
5         y_1 := y;
6         z_1 := z;
7         r := delete(r,y_1 + z_1);
8         z := z − 1;
9       y_2 := y;
10      z_2 := z;
11      r := insert(r,y_2 + z_2);
12      y := y − 1;
13    y_3 := y;
14    z_3 := z;
15    r := insert(r,y_3 + z_3);
```



The graph annotations for the assignment to a state variable nodes.

node	*effects*	*delta-condition*	*possible*	*δPossible*
2	$\{\}$	$\{\}$	$\{\}$	$\{\}$
15	$\{x_1 \in r\}$	$\{x_1 \neq y_1 + z_1\}$	$\{18\}$	$\{\langle 18,\ y_1 + z_1 \neq y_2 + z_2\rangle\}$
18	$\{x_1 \in r\}$	$\{\}$	$\{15\}$	$\{\langle 15,\ y_1 + z_1 \neq y_2 + z_2\rangle\}$
14	$\{x_1 \in r\}$	$\{\}$	$\{15\}$	$\{\langle 15,\ y_1 + z_1 \neq y_3 + z_3\rangle\}$

□

4.2.2 Code Transformation

The full transformation for *While* updates extends the previous one for *While* updates without loops. As before, it interleaves within the update code commands for navigation of the computation graph (using the iterator functions *node*(), *right*(), *left*()) and run time testing of effect preservation (using *getDelta*(*node*()) and a procedure *test_δPossible*(*node*())). In addition, the transformation adds manipulation of run time value recording, using a procedure *updateValues*(*node*(), $\langle x_1, \ldots, x_m \rangle$), that records in the given node the current values of the variables $x_1, \ldots, x_m$.

The *updateValues* procedure handles the run time recording of variable values within repeated loop rounds. For that purpose, a node n in the graph that represents an assignment $r := e(r, x_1, \ldots, x_m)$ to a state variable r (e.g., e = insert or delete), is associated at run time with a collection *values*(n) of tuples of values of $x_1, \ldots, x_m$ used in the expression e. Whenever the assignment is executed, the

procedure $updateValues(n, x_1, \ldots, x_m)$ is applied, and adds the current values of the variables $x_1, \ldots, x_m$ to the collection.

The *test_δPossible* procedure applies a run time test for the $\delta Possible$ conditions that annotate the node n (the delta conditions are a compile time product). Recall that

$$\delta Possible(n) = \{\langle m,\ \delta_m\rangle \mid \delta_m = \delta(label(n), effects(label(m))),\ m \text{ is an assignment to a state variable node, residing on a cycle through } n\}.$$

For each pair $\langle m,\ \delta_m\rangle$ in $\delta Possible(n)$, $test\_\delta Possible(n)$ applies the associated delta-condition δ_m, when instantiated by $values(m)$. This way the delta-conditions between the assignment of n and the effects of all previous executions of the assignment in m are tested. Assume that $values(m)$ records the values of variables $x_1, \ldots, x_m$.

$test\_\delta Possible(n) =$
For every $\langle m, \delta_m\rangle \in \delta Possible(n)$ such that $\delta_m \not\equiv \{\}$:
For every entry $v_1, \ldots, v_m$ in $values(m)$:
Substitute in δ_m: $x_1/v_1, \ldots, x_m/v_m$, and apply δ_m.
$test\_\delta Possible(n) = true$ if all tests are $true$ and $false$ otherwise.

The $reviseUpdate_2$ algorithm extends the previous $reviseUpdate_1$ by modifying the assignment to state variable transformation and adding a fourth transformation for a *while* statement. We list only these extensions:

Algorithm 7. [$reviseUpdate_2$ – **Code transformation for** $While$]
input: *A While update S.*
output: *An extended While update which is a minimal-EP restriction of S, when combined with the computation graph of S .*
method:

1.*a.* $S = [r := e(r, x_1, \ldots, x_m)]$, *an assignment to a state variable r, where expression e is over variables* $x_1, \ldots, x_m$*:*
Replace S by
[if getDelta(node()) and test_δPossible(node())
then $updateValues(node(), \langle x_1, \ldots, x_m\rangle); S; left()$
else fail];
4. S *=[while P do S']: Replace S by:*
[while P do { left();left();] · $reviseUpdate_2(S')$· *[}; right()]*

Brackets { and } serve to enclose a block of statements. The double `left();left()` *for loops is required to jump over the added* `if-then-else` *node.* □

Example 11. (The EP_2 Transformation of a non effect preserving update into an effect preserving one.) For the update S in Example 10, algorithm 5 returns the following update where lines are labelled accodring the orginal line of code:

```
1     x1 := x;
2a    if getDelta(node()) and test_δPossible(node()) then
2b       updateValues(node(),⟨x1⟩);
2c       r := insert(r, x1); left()
2d    else fail ;
3a    while y > 0 do
3b       left(); left()
4a       while z > 0 do
4b          left();
5           y1 := y; left();
6           z1 := z; left();
7a          if getDelta(node()) and test_δPossible(node()) then
7b             updateValues(node(),⟨y1, z1⟩);
7c             r := delete(r, y1 + z1);left()
7d          else fail;
8           z := z − 1; left();
4c       right();
9        y2 := y; left();
10       z2 := z; left();
11a      if getDelta(node()) and test_δPossible(node()) then
11b         updateValues(node(),⟨y2, z2⟩);
11c         r := insert(r, y2 + z2); left()
11d      else fail;
12       y := y − 1; left();
3c    right();
13    y3 := y; left();
14    z3 := z; left();
15a   if getDelta(node()) and test_δPossible(node()) then
15b      updateValues(node(),⟨y3, z3⟩);
15c      r := insert(r, y3 + z3); left()
15d   else fail                                          □
```

4.3 Correctness and Complexity of the Update Transformations

4.3.1 Enforcing Effect Preservation on Execution Sequences

In this subsection we concentrate on effect preservation in the semantic domain of *While* programs, i.e., the set of all execution sequences over a given set of variables. This "pre-processing" study is essential since effect preservation properties of *While* updates are defined in terms of effect preservation of their execution sequences, and correctness of *While* transformations is proved by refering to their impact on their execution sequences. We introduce an execution sequence transformation that enforces *minimal effect preservation*. The transformation is further optimized by using *delta-conditions*. These transformations are used later on to prove the effect

preservation properties of the update transformations.

The Conditional Assignment Transformation

The following transform maps an execution sequence Ψ to an execution sequence denoted $CA(\Psi)$, which is a minimal effect preserving restriction of Ψ.

Definition 10. [Conditional assignment transformation]
Let $\Psi = \langle S_0, s_0\rangle, \langle S_0, s_0\rangle, \ldots$, be an execution sequence. $CA(\Psi)$ is the execution sequence resulting from the replacement of every assignment to a state variable configuration $\langle A; S, s_i\rangle$ in Ψ by the configuration sequence $\gamma_1, \gamma_2, \gamma_3$, where:
$\gamma_1 = \langle A;\ if\ \neg effects_i(\Psi)\ then\ fail\ else\ skip; S,\ s_i\rangle$,
$\gamma_2 = \langle if\ \neg effects_i(\Psi)\ then\ fail\ else\ skip; S,\ s_{i+1}\rangle$,
$\gamma_3 = \begin{cases} \langle fail,\ s_{i+1}\rangle & \text{if } s_{i+1} \not\models effects_i(\Psi) \\ \langle skip; V,\ s_{i+1}\rangle & \text{otherwise} \end{cases}$
If the resulting sequence includes a failing configuration cut the sequence after the first failing configuration.

Example 12 (The CA Transformation). Consider the execution sequence from Example 3

$$\langle r := insert(r,x); r := delete(r,y),\ s\rangle \Rightarrow \langle r := delete(r,y),\ s'\rangle \Rightarrow \langle \epsilon,\ s''\rangle$$

and states s, s', s'' where $s' = s[r \mapsto insert(r,x)^s]$ and $s'' = s'[r \mapsto delete(r,y)^{s'}]$. The effects of the configurations in the sequence are $effects_0(\Psi) = \{\}$, $effects_1(\Psi) = \{x \in r\}$ and $effects_2(\Psi) = \{x \in r, y \notin r\}$.

1. Assume: $s = [r \mapsto \{3,4,5\}, x \mapsto 3, y \mapsto 4]$. Then, $CA(\Psi)$ is[4]

$\langle r := insert(r,x);\ if\ true\ then\ fail\ else\ skip;\ r := delete(r,y),\ s\rangle \Rightarrow$
$\langle if\ true\ then\ fail\ else\ skip;\ r := delete(r,y),\ s'\rangle \Rightarrow$
$\langle skip;\ r := delete(r,y),\ s'\rangle \Rightarrow$
$\langle r := delete(r,y);\ if\ x \notin r\ then\ fail\ else\ skip,\ s'\rangle \Rightarrow$
$\langle if\ x \notin r\ then\ fail\ else\ skip,\ s''\rangle \Rightarrow$
$\langle skip,\ s''\rangle \Rightarrow$
$\langle \epsilon,\ s''\rangle$

2. Assume: $s = [r \mapsto \{3,4,5\}, x \mapsto 3, y \mapsto 3]$. Then, $CA(\Psi)$ is

$\langle r := insert(r,x);\ if\ true\ then\ fail\ else\ skip;\ r := delete(r,y),\ s\rangle \Rightarrow$
$\langle if\ true\ then\ fail\ else\ skip;\ r := delete(r,y),\ s'\rangle \Rightarrow$
$\langle skip;\ r := delete(r,y),\ s'\rangle \Rightarrow$
$\langle r := delete(r,y);\ if\ x \notin r\ then\ fail\ else\ skip,\ s'\rangle \Rightarrow$
$\langle if\ x \notin r\ then\ fail\ else\ skip,\ s''\rangle \Rightarrow$
$\langle fail,\ s''\rangle$

[4] An empty effects set is the formula *true*.

That is, for the state $s = [r \mapsto \{3,4,5\}, x \mapsto 3, y \mapsto 4]$, in which Ψ is effect preserving (see Example 4), $CA(\Psi)$ is also effect preserving, successful and ends in the same state, while for the state $s = [r \mapsto \{3,4,5\}, x \mapsto 3, y \mapsto 3]$, in which Ψ is not effect preserving, $CA(\Psi)$ is failing. In any case, for both states, $CA(\Psi) \leq_{EP} \Psi$.

The following claim shows that the CA transformation does not needlessly restrict execution sequences that are already effect preserving:

Claim 1. [Correctness and Minimality of the CA transformation]
For every execution sequence Ψ, $CA(\Psi) \leq_{min\_EP} \Psi$.

Therefore, we conclude that the CA transformation does not needlessly restrict execution sequences that are already effect preserving. Such transformations are termed *minimal effect preserving transformations.*

Conclusion 1. *The CA transformation is a* minimal effect preserving transformation. *That is, $CA(\Psi) \leq_{EP} \Psi$, and if Ψ is already effect preserving, then $CA(\Psi) \equiv \Psi$.*

The Delta-Conditions Based Transformation

Delta-conditions were introduced as minimal conditions for guaranteeing that the effects of an intermediate configuration in an execution sequence are preserved by its successor configuration. Therefore, the CA transformation can be revised into a delta-conditions based transformation CA_δ that replaces tests of full effects by tests of delta-conditions. Of course, the revised transformation CA_δ must preserve the properties of the former CA transformation.

Definition 11. [Delta-conditions based transformation]
Let $\Psi = \langle S_0, s_0\rangle, \ldots, \langle S_k, s_k\rangle$ be an execution sequence. $CA_\delta(\Psi)$ is the execution sequence resulting from the replacement of every assignment to a state variable configuration $\langle A; S, s_i\rangle$ in Ψ, by the configuration sequence γ_1, γ_2, where:
$\gamma_1 = \langle if\ \delta_i(\Psi)\ then\ A\ else\ fail; S,\ s_i\rangle$,
$\gamma_2 = \begin{cases} \langle A; S,\ s_i\rangle & if\ s_i \models \delta_i(\Psi) \\ \langle fail,\ s_i\rangle & otherwise \end{cases}$
If the resulting sequence includes a failing configuration cut the sequence after the first failing configuration.

Example 13 (The delta-conditions based Transformation). Consider the execution sequence from Example 3, and states s, s', s'', where $s' = s[r \mapsto insert(r,x)^s]$ and $s'' = s'[r \mapsto delete(r,y)^{s'}]$. The delta-conditions of the configurations in the sequence are $\delta_0(\Psi) = \{\}$, $\delta_1(\Psi) = \{x \neq y\}$ and $\delta_2(\Psi) = \{\}$.

1. Assume: $s = [r \mapsto \{3,4,5\}, x \mapsto 3, y \mapsto 4]$. Then, $CA_\delta(\Psi)$ is

$$\begin{array}{l} \langle if\ true\ then\ r := insert(r,x)\ else\ fail;\ r := delete(r,y),\ s\rangle \Rightarrow \\ \langle r := insert(r,x);\ r := delete(r,y),\ s\rangle \Rightarrow \\ \langle if\ x \neq y\ then\ r := delete(r,y)\ else\ fail,\ s'\rangle \Rightarrow \\ \langle r := delete(r,y),\ s'\rangle \Rightarrow \\ \langle \epsilon,\ s''\rangle \end{array}$$

2. Assume: $s = [r \mapsto \{3,4,5\}, x \mapsto 3, y \mapsto 3]$. Then, $CA_\delta(\Psi)$ is

$$\langle if\ true\ then\ r := insert(r,x)\ else\ fail;\ r := delete(r,y),\ s\rangle \Rightarrow$$
$$\langle r := insert(r,x);\ r := delete(r,y),\ s\rangle \Rightarrow$$
$$\langle if\ x \neq y\ then\ r := delete(r,y)\ else\ fail,\ s'\rangle \Rightarrow$$
$$\langle fail,\ s'\rangle \Rightarrow$$

Compared with the execution sequence that results from the CA-transformation in Example 12, the execution sequence $CA_\delta(\Psi)$ is shorter, and there is a single equality test which precedes the assignment. As before, for both states, $CA_{delta}(\Psi) \leq_{EP} \Psi$.

Claim 2. [Correctness and Minimality of the CA_δ transformation] *For every execution sequence Ψ, $CA_\delta(\Psi) \leq_{min\_EP} \Psi$.*

Conclusion 2. *The CA_δ transformation is a* minimal effect preserving transformation. *That is, $CA_\delta(\Psi) \leq_{EP} \Psi$, and if Ψ is already effect preserving, then $CA_\delta(\Psi) \equiv \Psi$.*

4.3.2 Minimal Effect Preserving Update Transformation

In this subsection we prove that the update transformations introduced above are minimal effect preserving. Recall that a transformation Θ is minimal effect preserving, if it produces minimal-EP-restrictions of its input updates (Definition 7). Observing the definition of this relation, it means that for all states s, $seq(\Theta(U), s) \leq_{min\_EP} seq(U, s)$ (Definition 6). Our proof uses the results about the CA_δ transformation of execution sequences. For each of the two update transformations Θ introduced in Subsections 4.1 and 4.2, we show that

for every update U, for all states s, $seq(\Theta(U), s) \equiv_{EP} CA_\delta(seq(U, s))$.

Since by Proposition 2:

for every update U, for all states s, $CA_\delta(seq(U, s)) \leq_{min\_EP} seq(U, s)$,

it follows that

for every update U, for all states s, $seq(\Theta(U), s) \leq_{min\_EP} seq(U, s)$.

Therefore, by Definition 6,

for every update U, $\Theta(U) \leq_{min\_EP} U$,

and by Definition 7, Θ is minimal effect preserving.

Correctness of the transformation of *While* updates without loops

The main problem is to show

for every update U, for all states s, $seq(reviseUpdate_1(U), s) \equiv_{EP} CA_\delta(seq(U, s))$.

Once this is proved, the minimal effect preservation property of EP_1 is obtained as outlined above.

Lemma 1. *Let U be a While update without loops, and s a state. The execution sequence $seq(U,s)$ corresponds to a full path in the computation tree of U, $CT(U)$, such that:*

1. *$seq(U,s)_0$ corresponds to $root(CT(U))$, and if $seq(U,s)_i$ corresponds to node n_i in the tree, then $seq(U,s)_{i+1}$ corresponds to a child node n_{i+1} of n_i. If $seq(U,s)$ is a successful sequence, the last terminal configuration does not correspond to any node, and its previous configuration corresponds to a leaf node. If the sequence is failing, its last configuration corresponds to a leaf node.*
   This correspondence defines a $1:1$ mapping between the configurations in $seq(U,s)$ (excluding the terminal configuration, if exists) and the nodes of the path.

2. *If $seq(U,s)_i$ corresponds to node n_i, then $effects_i(seq(U,s)) = effects(n_i)$ and $\delta_i(seq(U,s)) = \delta(n_i)$.*

Proposition 4. *Let U be an extended While update, such that all external calls are to terminating procedures. Let $removeEC(U)$ be the While update that is obtained from U by removing all external calls (if there is a syntactic problem, an external call is replaced by skip). Then, for every state s: $seq(removeEC(U),s) \equiv_{EP} seq(U,s)$.*

Proof. The external calls do not affect termination and any variable assignment. Therefore, for every state s, $seq(removeEC(U),s)$ agrees with $seq(U,s)$ with respect to termination and failures, and if $seq(U,s)$ is effect preserving so is $seq(removeEC(U),s)$. □

Lemma 2. *Let U be a While update without loops, and s a state. Then,*

$$removeEC(seq(reviseUpdate_1(U),s)) = CA_\delta(seq(U,s)).$$

Theorem 1. [Correctness and Minimality of Algorithm 4] *For every update S in While without loops, $reviseUpdate_1(S)$ is a minimal EP restriction of S.*

Proof. By Lemma 2, $(seq(removeEC(reviseUpdate_1(U)),s)) = CA_\delta(seq(U,s))$. By Proposition 4, for every update U that does not include non-terminating external calls, for every state s: $seq(removeEC(U),s) \equiv_{EP} seq(U,s)$. Therefore, $seq(reviseUpdate_1(U),s) \equiv_{EP} CA_\delta(seq(U,s))$. The rest of the proof is as outlined above. □

The following claim holds under the experimental observation that the size of delta-conditions is small, and is independent from the length of the execution sequence. The reason is that multiple contradictory assignments to the same relation in a single execution sequence do not happen frequently.

Claim 3. *The run time overhead of* $reviseUpdate_1(S)$ *is* $\mathcal{O}(size(S))$.

Proof. By the last theorem, $reviseUpdate_1(S) \leq_{min\_EP} S$. Therefore, for every state s, $seq(reviseUpdate_1(S), s)$ is finite. For every state s, the overhead of $seq(reviseUpdate_1(S), s)$ over $seq(S, s)$ is

$$length(seq(S, s)) \times (Time(getDelta(node())) + Time(navigation\ procedures))$$

Since there are no loops, the length of $seq(S, s)$ is bounded by the $size(S)$.

$$Time(getDelta(node())) = size(delta\text{-}condition) \times Time(condition\ test).$$

Under the above assumption, the size of the delta-conditions is bounded. Each condition in a delta-condition is a variable inequality, and therefore, its test takes constant time, since it does not depend on the size of a relation. Therefore, $Time(getDelta(node())) = \mathcal{O}(\infty)$. Navigation procedures are $calO(1)$, since they only advance the iterator. Therefore, the overall overhead is $\mathcal{O}(size(S))$. □

Correctness of the transformation of *While* updates

The correctness of the $reviseUpdate_2$ transformation is proved, essentially, similarly to the proof for $reviseUpdate_1$. However, there are two tricky points:

1. *While* allows for infinite execution sequences.

2. A node in the computation graph of a *While* update is annotated with delta-conditions with respect to all possible assignments that might precede its statement when executed. Therefore, its set of delta-condition is a superset of the actual delta-conditions of its corresponding configuration in an execution sequence.

Therefore, the two Lemmas on which the correctness Theorem is based are slightly different.

Lemma 3. *Let U be a While update, and s a state. Every finite prefix of $seq(U, s)$ corresponds to a path from the root in the computation graph of U, $CG(U)$, such that:*

1. *$seq(U, s)_0$ corresponds to $root(CG(U))$, and if $seq(U, s)_i$ corresponds to node n_i in the tree, then $seq(U, s)_{i+1}$ corresponds to a child node n_{i+1} of n_i. If $seq(U, s)$ is a finite successful sequence, the last terminal configuration does not correspond to any node, and its previous configuration corresponds to a leaf node. If the sequence is finite failing, its last configuration corresponds to a leaf node.*
   This correspondence defines a partial mapping from configurations in $seq(U, s)$ (excluding the terminal configuration, if exists) to the nodes on the path.

2. *If* $seq(U,s)_i$ *corresponds to node* n_i*, then* $effects_i(seq(U,s)) = effects(n_i)$ *and* $\delta_i(seq(U,s)) \subseteq \delta(n_i) \cup \delta Possible(n_i)$[5].

Lemma 4. *Let* U *be a While update, and* s *a state. Then,*

$$removeEC(seq(reviseUpdate_2(U), s)) \equiv_{EP} CA_\delta(seq(U,s)).$$

Theorem 2. [Correctness and Minimality of Algorithm 7] *For every update* S *in While,* $reviseUpdate_2(S)$ *is a minimal EP restriction of* S.

Proof. By Lemma 4, $(seq(removeEC(reviseUpdate_1(U)), s)) \equiv_{EP} CA_\delta(seq(U,s))$. By Proposition 4, for every update U that does not include non-terminating external calls, for every state s: $seq(removeEC(U), s) \equiv_{EP} seq(U,s)$. Therefore, $seq(reviseUpdate_1(U), s) \equiv_{EP} CA_\delta(seq(U,s))$. The rest of the proof is as outlined above. □

As before, the complexity claim holds under the experimental observation that the size of delta-conditions is small, and is independent from the length of the execution sequence.

Claim 4. *If for a state* s $seq(S,s)$ *is terminating, then the run time overhead of* $seq(reviseUpdate_2(S), s)$ *is proportional to the multiplication of the length of the execution sequence* $seq(S,s)$ *by the number of loop repetitions.*

Proof. By the last theorem, $reviseUpdate_2(S) \leq_{min\_EP} S$. Therefore, if $seq(S,s)$ is finite, then also $seq(reviseUpdate_2(S), s)$ is finite. For every state s, the overhead of $seq(reviseUpdate_2(S), s)$ over $seq(S,s)$ is

$$length(seq(S,s)) \times (\quad Time(getDelta(node())) + Time(test\_\delta Possible(node())) + Time(updateValues(node())) + Time(navigation\ procedures))$$

As in the no-loops case, $Time(getDelta(node())) = \mathcal{O}(1)$, and navigation procedures are $\mathcal{O}(1)$, since they only advance the iterator. The procedure $updateValues(node(), x_1, \ldots, x_m)$ is also $\mathcal{O}(1)$, since it adds a new value to a collection. The only additional complexity overhead results from the procedure $test\_\delta Possible(node())$ that tests multiple tuples that record variable values resulting from loop repetitions. A call to $test\_\delta Possible(node())$ takes time proportional to the number of loop repetitions. Therefore, the run time overhead is $\mathcal{O}(length(seq(S,s)) \times \#(\text{loop repetitions}))$. The number of loop repetitions depends on the update arguments, and usually is much smaller than the the length of the execution sequence. Therefore, the run time overhead of $seq(reviseUpdate_2(S), s)$ is much smaller than $\mathcal{O}((length(seq(S,s))^2))$, which is the overhead of a purely run time effect preservation procedure. □

[5] The notation is used imprecisly here, since $\delta Possible(n_i)$ is a set of pairs, and only the second element in each pair is a delta-condition.

5 Related Works

Effect preservation is traditionally related to the paradigm of *integrity constraint management*. This direction is relevant in dynamic situations, where operations can violate necessary properties. The role of integrity constraint maintenance is to guard the consistency of the information base. There are, basically, two major approaches to maintain consistency: *Integrity checking*, where operations are tested, either at run time or at compile time, for being integrity preserving [10, 9, 8, 11, 21, 7], and *integrity enforcement*, where operations are *repaired*, so to guarantee consistency [37, 28, 12, 29, 23, 14, 24, 3, 4]. The problem of effect violation arises in the latter approach, where transactions are automatically constructed. It can also arise in situations where transactions are deductively synthesized ([27]). The *Greatest Consistent Specialization* theory of [35] is a compile time enforcement theory with effect preservation. It generates a fully repaired transaction, which is consistent with respect to a given set of constraints, and preserves the effects of the original transaction. A relaxed version, that allows for stronger repairs, is suggested in [30, 22]. A classification of Research efforts in consistency enforcement appears in [25].

The most common approach in integrity enforcement is that of the run time *Rule Triggering Systems (RTS)* (*active databases*), which are implemented in almost every commercial database product [38, 37, 16, 38]. RTSs have to cope with problems of *termination, uniqueness* (*confluence*) and *effect preservation* [12, 13, 36, 39]. Since a rule might be repeatedly applied, an application of a set of rules does not necessarily terminate. Practically, the problem is solved by timing-out or counting the level of nested rule activations. Sophisticated methods [12, 13] deal with static cycle analysis of rules. Furthermore, different orders of rule execution do not guarantee a unique database state. This is the *confluence* problem of RTSs. Static analysis of confluence in RTSs is studied in [36, 39].

The problem of effect violation in active databases occurs when a rule fires other rules that perform an update that is contradictory to an update performed by a previous rule. Automated generation and static analysis of active database rules [37, 6, 12] do neither handle, nor provide a solution to that problem. In fact, contradicting updates are allowed, since the case of inserting and deleting the same tuple is considered as a *skip* operation. The problem lies in the locality of rule actions. Since an action is intended to repair a single event, it does not account for the history of the repair, and therefore might not preserve the original intention of the transaction being repaired. The limitations of RTSs in handling effect preservation are studied in [33, 34].

In earlier versions of this work [17, 1, 2, 18], effect preservation was syntactically defined, based on the data structure that is constructed for a transaction. [17] presents an early investigation of the usage of dependency graphs for ordering constraint enforcement. [1, 2] present our first effect preservation efforts for *While* statements without loops: First, sequence transactions were annotated with two kinds of effects (*desired* and *executed*), and a sequence was said to be effect preserving if its desired effects logically followed from its executed effects. Then,

a loop-less transaction was said to be effect preserving if all sequence transactions on its computation tree are effect preserving. However, in [18] we tried to extend this approach for handling effect preservation in general *While* transactions, but encountered major problems, since the meaning of the desired and executed effects in the presence of loops is not straightforward. Consequently, we changed our overall approach to effect preservation, looking for a unified framework that can account for all *While* transactions. The result is a new approach, were effect preservation is semantically defined (rather than syntactically). The move from syntax based effect preservation to semantics based one is dramatic, since the latter provides a uniform framework for *While*, on which general criteria for effect preserving transformations can be developed. The effect preservation terminology developed in Section 3, and the algorithms introduced in section 4 could not have been developed on the basis of syntax based effect preservation alone.

6 Conclusion

In this work we introduced a combined, compile time – run time method for enforcing effect preservation in rule triggering systems. Our method enforces effect preservation on updates written in an imperative language with loops. It is based on the assumption that *effects of primitive database updates* are provided by the developer. The transformation is proved to be *minimal effect preserving*, and under certain conditions provides meaningful improvement over the quadratic overhead of pure run time procedures.

Our goal is to produce a database tool in which effect preservation is a correctness condition for transactions. For that purpose, we are currently implementing our effect preservation algorithm by embedding it within a real database platform. We intend to use an open source database, such as postgres, and apply effect preservation to stored procedures (procedural database application code maintained within the database). Moreover, we plan to extend the effect preservation process so that it will be applicable to general transactions (and not only to isolated primitive updates). In addition, we plan to experimentally test the assumption about the independence of the size of delta-conditions from the length of the execution sequence.

Further research is needed in order to extend the set of primitive updates such that it includes, for example, attribute modification. Study of dynamic constraints requires further research as well. Another future application domain is semi-structured databases. The theory applies to any domain, provided that the developer associates effects with primitive assignments, and provides an algorithm for deriving delta-conditions.

Acknowledgments: We would like to thank B. Thalheim, K.D. Schewe, F. Bry, E. Mayol, A. Cali and M. Kifer for providing constructive comments on our work.

References

[1] Balaban, M. and Jurk, S. Intentions of Operations – Characterization and Preservation. In *Proc. International ER'02 workshop on Evolution and Change in Data Management (ECDM'02)*, pages 100–111, 2002.

[2] Balaban, M. and Jurk, S. Update-Consistent Query Results by Means of Effect Preservation. In *Proc. Fifth International Conf. on Flexible Query Abswering Systems (FQAS'02)*, pages 28–43, 2002.

[3] Balaban, M. and Shoval, P. Enhancing the ER model with structure methods. *Journal of Database Management*, 10(4), 1999.

[4] Balaban, M. and Shoval, P. MEER – an EER model enhanced with structure methods. *Information Systems Journal*, pages 245 – 275, 2001.

[5] Baralis, E. and Widom, J. An algebraic approach to rule analysis in expert database systems. *Proceedings of the 20. International Conference on Very Large Data Bases*, 1990.

[6] Baralis, E. and Widom, J. An algebraic approach to static analysis of active database rules. In *ACM Transactions on Database Systems*, volume 25(3), pages 269–332, September 2000.

[7] Benzaken, V. and Schaefer, X. Static Integrity Constraint Management in Object-Oriented Databases Programming Languages via Predicate Transformers. In *European Conference on Object-Oriented Programming, ECOOP'97*, Lecture Notes in Computer Science, 1997.

[8] Benzaken, V. and Themis, D. A database programming language handling integrity constraints. *VLDB Journal*, pages 493 – 518, 1995.

[9] Bry, F. Intensional updates: Abduction via deduction. In *Proc. 7th Conf. on Logi Programming*, 1990.

[10] Bry, F. and Manthey, R. Checking consistency of database constraints: A logical basis. In *Proc. of the VLDB int. Conf.*, pages 13–20, 1986.

[11] Celma, M. and Decker, H. Integrity checking in deductive databases. the ultimate method? *Proceedings of 5th Australiasian Database Conference*, pages 136–146, 1995.

[12] Ceri, S., P.Fraternali, Paraboschia, S., and Tanca, L. Automatic gerneration of production rules for integrity maintenance. In *ACM Transactions on Database Systems*, volume 19(3), pages 367–422, 1994.

[13] Ceri, S. and Widom, J. Deriving production rules for constraint maintenance. *Proceedings of the 16. International Conference on Very Large Data Bases*, pages 566–577, 1990.

[14] Etzion, O. and Dahav, B. Patterns of self-stabilization in database consistency maintenance. *Data and Knowledge Engineering*, 28(3):299–319, 1998.

[15] Fikes, R.E. and Nilsson, N.J. Strips: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2:189–208, 1971.

[16] Fraternali, P., Paraboschi, S., and Tanca, L. Automatic rule generation for constraints enforcement in active databases. In Lipeck, U. and Thalheim, B., editors, *Modeling Database Dynamics*, pages 153–173. springer WICS, 1993.

[17] Jurk, S. and Balaban, M. Improving Integrity Constraint Enforcement by Extended Rules and Dependency Graphs. In *Proc. 22th Conf. on DEXA*, 2001.

[18] Jurk, S. and Balaban, M. Towards Effect Preservation of Updates with Loops. In *Proc. Fifth IFIP TC-11 WG 11.5 Working Conf. on Integrity and Internal Control in Information Systems (IICIS'02)*, pages 59–75, 2002.

[19] Kniesel, G. *ConTraCT - A Refactoring Editor Based on Composable Conditional Program Transformations.* Technical Report, Computer Science Dept., University of Bonn, 2005.

[20] Kniesel, G. and Koch, H. Static composition of refactorings. *Science of Computer Programming, Special Issue on "Program Transformation", Lammel, R. (ed.)*, 52:9–51, 2004.

[21] Lee, S.Y. and Ling, T.W. Further improvement on integrity constraint checking for stratisfiable deductive databases. In *Proc. 22th Conf. on VLDB*, pages 495–505, 1996.

[22] Link, S. Consistency enforcment in databases. In Bertossi, L., Katona, G.O.H., Schewe, K.-D., and Thalheim, B., editors, *Semantics in Databases, Second International Workshop, Dagstuhl Castle, Germany*, 2003.

[23] Mayol, E. and Teniente, E. Structuring the process of integrity maintenance. In *Proc. 8th Conf. on Database and Expert Systems Applications*, pages 262–275, 1997.

[24] Mayol, E. and Teniente, E. Addressing efficiency issues during the process of integrity maintenance. In *Proc. 10th Conf. on Database and Expert Systems Applications*, pages 270–281, 1999.

[25] Mayol, E. and Teniete, Ernest. A survey of current methods for integrity constraint maintenance and view updating. In Chen, Embley, Kouloumdjian, Liddle, Roddick, editor, *Intl. Conf. on Entity-Relationship Approach*, volume 1727 of *Lecture Notes in Computer Science*, pages 62–73, 1999.

[26] Nielson, H.R. and Nielson, F. *Semantics with Applications - A Formal Introduction.* John Wiley & Sons, 1992.

[27] Qian, X. The deductive synthesis of database transactions. *ACM Transactions on Database Systems*, pages 626 – 677, 1993.

[28] Qian, X., Jullig, R., and Daum, M. Consistency Management in a Project Management Assisteant. In *ACM-SIGSOFT'90*, 15(6), 1990.

[29] Ross, K.A. and Srivastava, D. Materialized View Maintenance and Integrity Constraint Cheking: Trading Space for Time. In *ACS-SIGMODF'96*, 1996.

[30] S. Link, K.D. Schewe. Towards an arithmetic theory of consistency enforcement based on preservation of δ constraints. In *Electronic Notes in Theoretical Computer Science*, volume 61, pages 1–20, 2002.

[31] Sacerdoti, E. The nonlinear nature of plans. In *ijcai-75*, pages 206–214, 1975.

[32] Schewe, K.D. Consistency enforcement in entity-relationship and object-oriented models. *Data and Knowledge Eng.*, 28(1):121–140, 1998.

[33] Schewe, K.D. and Thalheim, B. Consistency enforcement in active databases. In Chakravarty, S. and Widom, J., editors, *Research Issues in Data Engineering - Active Databases*, pages 71–76. IEEE Computer Society Press, 1994.

[34] Schewe, K.D. and Thalheim, B. Limitations of rule triggering systems for integrity maintenance in the context of transition specifications. *Acta Cybernetica*, 13:277–304, 1998.

[35] Schewe, K.D. and Thalheim, B. Towards a theory of consistency enforcement. *Acta Informatics*, 36:97–141, 1999.

[36] van der Voort and Siebes, A. Termination and confluence of rule execution. In *In Proceedings of the Second International Conference on Information and Knowledge Management*,. November 1993.

[37] Widom, J. and Ceri, S. Deriving production rules for constraint maintenance. In *Proc. 16th Conf. on VLDB*, pages 566–577, 1990.

[38] Widom, J. and Ceri, S. *Active Database Systems.* Morgan-Kaufmann, 1996.

[39] Zhou, C. and Hsu, M. A theory for rule triggering systems. In *Advances in Database Technology-EDBT' 90*, volume 416 of *Lecture Notes in Computer Science*, pages 407–421, 1999.

7 Appendix – Proofs

Proof of Proposition 1:

Proposition: An execution sequence $\Psi = \langle S_0, s_0\rangle, \langle S_1, s_1\rangle, \ldots$ in which $s_{i+1} \models \mathit{effects}_i(\Psi)$ for all $i \geq 0$, is effect preserving.
Proof: This property derives from the necessary property of effects of assignments

A: For every state s, if $\langle A, s\rangle \Rightarrow s'$ then $s' \models \mathit{effects}(A)$. By the definition of the effects of configurations in an execution sequence Ψ, if the statement that is executed in the i-th transition is not an assignment, then $\mathit{effects}_{i+1}(\Psi) = \mathit{effects}_i(\Psi)$, and if it is an assignment A, then $\mathit{effects}_{i+1}(\Psi) = \mathit{effects}_i(\Psi) \cup \mathit{effects}(A)$. Therefore, in the first case we have $s_{i+1} \models \mathit{effects}_i(\Psi) = \mathit{effects}_{i+1}(\Psi)$ and in the second case we have $s_{i+1} \models \mathit{effects}_i(\Psi)$ and $s_{i+1} \models \mathit{effects}(A)$ which implies $s_{i+1} \models \mathit{effects}_i(\Psi) \cup \mathit{effects}(A) = \mathit{effects}_{i+1}(\Psi)$. Since for $i = 0$, $s_0 \models \{\mathit{true}\} = \mathit{effects}_0(\Psi)$, we have the effect preservation property for all configurations in the sequence.

□

Proof of Claim 1:

Claim: For every execution sequence Ψ, $CA(\Psi) \leq_{\mathit{min\_EP}} \Psi$.
Proof: We have to show that $CA(\Psi)$ is EP and a minimal restriction of Ψ.

1. $CA(\Psi) \leq \Psi$: If $CA(\Psi)$ is failing then it is a restriction of Ψ. Otherwise, the sequence of states in $CA(\Psi)$ is the same as in Ψ (apart from possible intermediate repetitions). Therefore, if $CA(\Psi)$ is infinite, then also Ψ is infinite. If $CA(\Psi)$ is finite and successful then $\mathit{end}(CA(\Psi)) = \mathit{end}(\Psi)$.

2. $CA(\Psi)$ is EP: We show by induction on the sequence of states in $CA(\Psi)$ $s_0, s_1, \ldots$ that if the sequence is not failing, then for every state s_i, $s_i \models \mathit{effects}_i(CA(\Psi))$. First, we note that not only the states in $CA(\Psi)$ are states that occur in Ψ and in the same ordering, but also the effects associated with configurations in $CA(\Psi)$ are the same since $CA(\Psi)$ has no additional assignments. Therefore, for every $CA(\Psi)$ configuration there is a corresponding earlier Ψ configuration with the same effects.
   Basis: $s_0 \models \{\mathit{true}\} = \mathit{effects}_0(CA(\Psi))$.
   Inductive step: Assume that the claim holds for all states s_i, for $0 \leq i \leq k$, for some $k \geq 0$. Consider the transition $\langle W, s_k\rangle \Rightarrow \langle W', s_{k+1}\rangle$ in the sequence.

   a) If $\langle W, s_k\rangle$ is not an assignment configuration, then $s_k = s_{k+1}$ and $\mathit{effects}_k(CA(\Psi)) = \mathit{effects}_{k+1}(CA(\Psi))$, and by the inductive hypothesis: $s_{k+1} \models \mathit{effects}_{k+1}(CA(\Psi))$.

   b) If $\langle W, s_k\rangle$ is an assignment configuration, then

   $$W = A;\ \mathit{if}\ \neg\mathit{effects}_i(\Psi)\ \mathit{then}\ \mathit{fail}\ \mathit{else}\ \mathit{skip}; V,$$

   where the i configuration in Ψ corresponds to the k configuration in $CA(\Psi)$. That is, $\mathit{effects}_i(\Psi) = \mathit{effects}_k(CA(\Psi))$. The following configurations in the $CA(\Psi)$) sequence are
   $\langle \mathit{if}\ \neg\mathit{effects}_i(\Psi)\ \mathit{then}\ \mathit{fail}\ \mathit{else}\ \mathit{skip}; V,\ s_{k+1}\rangle \Rightarrow$
   either $\langle \mathit{fail},\ s_{k+1}\rangle$ if $s_{k+1} \not\models \mathit{effects}_i(\Psi)$,
   or $\langle \mathit{skip}; V,\ s_{k+1}\rangle$ if $s_{k+1} \models \mathit{effects}_i(\Psi)$.
   In the first case the sequence is failing, and hence EP. In the second case, $s_{k+1} \models \mathit{effects}_i(\Psi) = \mathit{effects}_k(CA(\Psi))$ and by Proposition 1, $s_{k+1} \models \mathit{effects}_{k+1}(CA(\Psi))$.

Therefore, in either case, $CA(\Psi)$ is EP.

3. $CA(\Psi) \leq_{min\_EP} \Psi$: It can be shown, by induction on the sequence configurations, that if Ψ is an EP execution sequence then $\Psi \leq_{EP} CA(\Psi)$.

□

Proof of Claim 2:

Claim: For every execution sequence Ψ, $CA_\delta(\Psi) \leq_{min\_EP} \Psi$.
Proof: The proof is similar to that of Proposition 1. It is based on the correspondence between configurations of $CA_\delta(\Psi)$ to those of Ψ: Ψ configurations that are not changed correspond to themselves, and assignment configurations in Ψ correspond to the pair of configurations in $CA_\delta(\Psi)$ that replaces them.

1. $CA_\delta(\Psi) \leq \Psi$ since it is either failing or follows the same configurations (with some additional intermediate ones).

2. In order to show that $CA_\delta(\Psi)$ is EP we show (by induction on the sequence of states in $CA_\delta(\Psi)$ $s_0, s_1, \ldots$) that if the sequence is not failing, then for every state s_k,
$s_k \models \mathit{effects}_k(CA_\delta(\Psi)) = \mathit{effects}_i(\Psi)$ (where the k-th configuration in $CA_\delta(\Psi)$ corresponds to the i-th configuration in Ψ).
Basis: $s_0 \models \{\mathit{true}\} = \mathit{effects}_0(CA_\delta(\Psi)) = \mathit{effects}_0(\Psi)$.
Inductive step: Assume that the claim holds for the first k configurations of $CA_\delta(\Psi)$, for some $k \geq 0$. Consider the transition $\langle W, s_k\rangle \Rightarrow \langle W', s_{k+1}\rangle$ in $CA_\delta(\Psi)$.

   a) If $\langle W, s_k\rangle$ is an original Ψ configuration, it is not an assignment configuration. Therefore $s_k = s_{k+1}$ and $\mathit{effects}_k(CA(\Psi)) = \mathit{effects}_{k+1}(CA(\Psi))$, and by the inductive hypothesis $s_{k+1} \models \mathit{effects}_{k+1}(CA(\Psi))$.

   b) If $\langle W, s_k\rangle$ is the first of a new pair of configurations that replaces the i-th assignment configuration in Ψ, then it is of the form
   $\langle \mathit{if}\ \delta_i(\Psi)\ \mathit{then}\ A\ \mathit{else}\ \mathit{fail}; V,\ s_k\rangle$,
   where the following $CA_\delta(\Psi)$ configurations are
   either $\langle A; V,\ s_{k+1}\rangle \Rightarrow \langle V,\ s_{k+2}\rangle$ (where $s_{k+1} = s_k$)
   or $\langle \mathit{fail},\ s_k\rangle$ (where $s_{k+1} = s_k$).
   In the latter case the overall $CA_\delta(\Psi)$ sequence is failing, and hence EP.
   In the first case, $s_{k+1} = s_k \models \delta_i(\Psi) = \delta(\mathit{effects}_i(\Psi), A)$. Since by the inductive hypothesis $s_{k+1} = s_k \models \mathit{effects}_i(\Psi)$, we have by the definition of delta-conditions: $s_{k+2} \models \mathit{effects}_i(\Psi)$. By the inductive hypothesis we also have $\mathit{effects}_i(\Psi) = \mathit{effects}_k(CA_\delta(\Psi)) = \mathit{effects}_{k+1}(CA_\delta(\Psi))$, since the k-th configuration is an if configuration. Altogether, from
   $s_{k+2} \models \mathit{effects}_{k+1}(CA_\delta(\Psi))$, and by Proposition 1, we get
   $s_{k+2} \models \mathit{effects}_{k+2}(CA_\delta(\Psi))$.

The second equality in the hypothesis is obtained directly from the definition of effects in execution sequence:

$$\begin{aligned} effects_{k+2}(CA_\delta(\Psi)) = \; & effects_{k+1}(CA_\delta(\Psi)) \cup effects(A) \; = \\ & effects_i(\Psi) \cup effects(A) \; = effects_{i+1}(\Psi). \end{aligned}$$

3. If Ψ is an EP execution sequence then $\Psi \leq_{EP} CA_\delta(\Psi)$. Can be shown by induction on the sequence configurations.

□

Proof of Lemma 1:

Lemma: Let U be a *While* update without loops, and s a state. The execution sequence $seq(U, s)$ corresponds to a full path in the computation tree of U, $CT(U)$, such that:

1. $seq(U, s)_0$ corresponds to $root(CT(U))$, and if $seq(U, s)_i$ corresponds to node n_i in the tree, then $seq(U, s)_{i+1}$ corresponds to a child node n_{i+1} of n_i. If $seq(U, s)$ is a successful sequence, the last terminal configuration does not correspond to any node, and its previous configuration corresponds to a leaf node. If the sequence is failing, its last configuration corresponds to a leaf node.
   This correspondence defines a $1 : 1$ mapping between the configurations in $seq(U, s)$ (excluding the terminal configuration, if exists) and the nodes of the path.

2. If $seq(U, s)_i$ corresponds to node n_i, then $effects_i(seq(U, s)) = effects(n_i)$ and $\delta_i(seq(U, s)) = \delta(n_i)$.

Proof: The proof is by induction on the structure of U (nesting level of its operators).

1. If U is primitive, then $seq(U, s)$ is either $\langle fail, s\rangle$ or $\langle U, s\rangle \Rightarrow s'$, and $CT(U)$ is a single node. So, the correspondence is established.

2. If U is an `if` statement `if` P `then` S_1 `else` S_2, then $seq(U, s) = \gamma_0, \ldots, \gamma_n$, where $\gamma_0 = \langle$`if` P `then` S_1 `else` $S_2, s\rangle$ and $\gamma_1, \ldots, \gamma_n$ is either $seq(S_1, s)$ or $seq(S_1, s)$. The root node of $CT(U)$ has two subtrees for $CT(S_1)$ and $CT(S_2)$. The inductive hypothesis holds for S_1 and S_2. Therefore, the tree path that corresponds to $seq(U, s)$ consists of $root(CT(U))$ and the path that corresponds to either S_1 or S_2, according to $seq(U, s)$.

3. If U is a sequence statement $S_1; S_2$, then $seq(U, s)$ is the concatenation of two sequences for S_1 and S_2, respectively. The inductive hypothesis holds for S_1 and S_2. $CT(U)$ is $CT(S_1)$ where all non $fail$ leaves have left subtrees for $CT(S_2)$. Therefore, the tree path that corresponds to $seq(U, s)$ consists of the path that corresponds to $seq(S_1, s)$ in $CT(S_1)$, concatenated to the path that corresponds to the S_2 sequence in $CT(S_2)$.

The second part of the Lemma holds since the definitions of effects and of delta-conditions for execution sequences are exactly the tree annotations.

□

Proof of Lemma 2:

Lemma: Let U be a *While* update without loops, and s a state. Then,

$$removeEC(seq(reviseUpdate_1(U), s)) = CA_\delta(seq(U, s)).$$

Proof: The proof is obtained from the following three immediate claims:

1. There is an order preserving correspondence (mapping) between the configurations of $seq(U, s)$ and the those of $CA_\delta(seq(U, s))$, such that:
   a) If $seq(U, s)_i$ is an assignment to a state variable configuration, then it corresponds to two successive configurations in $CA_\delta(seq(U, s))$ – an `if` configuration and either the assignment or a $fail$ configuration –, following the definition of the CA_δ transformation.
   b) All other configurations correspond to themselves.
2. There is an order preserving correspondence (mapping) between the configurations of $seq(U, s)$ and the those of $removeEC(seq(reviseUpdate_1(U), s))$, such that:
   a) If $seq(U, s)_i$ is an assignment to a state variable configuration, then it corresponds to two successive configurations in $removeEC(seq(reviseUpdate_1(U), s))$ – an `if` configuration and either the assignment or a $fail$ configuration –, following the definition of the $reviseUpdate_1$ transformation.
   b) All other configurations correspond to themselves.
3. When the sequence $removeEC(seq(reviseUpdate_1(U), s))$ reaches an `if` configuration that corresponds to an assignment to a state variable configuration $seq(U, s)_i$, the $node()$ iterator procedure points to the tree node n_i that corresponds to $seq(U, s)_i$.

Based on the first two claims, the two execution sequences $removeEC(seq(reviseUpdate_1(U), s))$ and $CA_\delta(seq(U, s))$ differ only in the conditions in the added `if` statements. In $CA_\delta(seq(U, s))$ the condition is $\delta_i(seq(U, s))$ – for the corresponding $seq(U, s)_i$ configuration, while in $removeEC(seq(reviseUpdate_1(U), s))$ the condition is $\delta(n_i)$ – for the tree node n_i pointed by the iterator. However, based on the third claim, the conditions are the same since $effects_i(seq(U, s)) = effects(n_i)$, by Lemma 1. Therefore, $removeEC(seq(reviseUpdate_1(U), s)) = CA_\delta(seq(U, s))$.

□

Proof of Lemma 3:

Lemma: Let U be a *While* update, and s a state. Every finite prefix of $seq(U,s)$ corresponds to a path from the root in the computation graph of U, $CG(U)$, such that:

1. $seq(U,s)_0$ corresponds to $root(CG(U))$, and if $seq(U,s)_i$ corresponds to node n_i in the tree, then $seq(U,s)_{i+1}$ corresponds to a child node n_{i+1} of n_i. If $seq(U,s)$ is a finite successful sequence, the last terminal configuration does not correspond to any node, and its previous configuration corresponds to a leaf node. If the sequence is finite failing, its last configuration corresponds to a leaf node.
   This correspondence defines a mapping from configurations in the finite prefix of $seq(U,s)$ (excluding the terminal configuration, if exists) to the nodes on the path.

2. If $seq(U,s)_i$ corresponds to node n_i, then $effects_i(seq(U,s)) = effects(n_i)$ and $\delta_i(seq(U,s)) \subseteq \delta(n_i) \cup \delta Possible(n_i)$[6].

Proof: We extend the proof of Lemma 1 by adding the additional entry for a *while* statement:

4. If U is a *while* statement while P do S, then a finite prefix of $seq(U,s)$ is a concatenation of repeating sequences for S:
   $\langle$while P do $S, s\rangle \Rightarrow \langle$`if` P `then` $(S;$ while P do $S)$ `else` $skip, s\rangle \cdot seq(S,s)$, where $\cdot$ stands for sequence concatenation. The root node of $CG(U)$ (a *while* labeled node) has a left child for the `if` statement, which has a left subtree for $CG(S)$. The inductive hypothesis holds for S. Therefore, the graph path that corresponds to a single round in the loop consists of $root(CG(U))$, its left child, and the path that corresponds to S in $CG(S)$. For the next round: The non failure leaves of $CG(S)$ have $root(seq(U,s)$ as their left child. Therefore, a path that corresponds to a finite prefix of $seq(U,s)$ consists of cyclic repetition on the graph path for a single loop round.

In the second part of the Lemma, the equality of effects holds since their definition for execution sequences is exactly the graph annotations. The delta-conditions and delta-Possible annotations of a graph node is a superset of the delta-conditions of the corresponding configuration since delta-Possible includes delta-conditions with respect to all possible assignments that might precede its statement when executed. Therefore, its set of delta-conditions is a superset of the actual delta-conditions of its corresponding configuration in an execution sequence.

□

[6] The notation is used imprecisely here, since $\delta Possible(n_i)$ is a set of pairs, and only the second element in each pair is a delta-condition.

Proof of Lemma 4:

Lemma: Let U be a *While* update, and s a state. Then,

$$removeEC(seq(reviseUpdate_2(U), s)) \equiv_{EP} CA_\delta(seq(U, s)).$$

Proof: The proof is the same as that of Lemma 2. The only difference is that the two sequences are not equal due to the difference in the delta-conditions in the added `if` statements. The conditions in the $removeEC(seq(reviseUpdate_2(U), s))$ sequence are taken from the graph-nodes – include the delta-conditions and delta-Possible annotations of a graph node, which form a superset of the delta-conditions of the corresponding configuration in $CA_\delta(seq(U, s))$. However, the tests of the conditions in both sequences give the same results since the extra delta-conditions in a configuration of $removeEC(seq(reviseUpdate_2(U), s))$ are evaluated on an empty set of variable values (they are applied on non-visited graph nodes, whose *values* collection is empty). Therefore, the two sequences, although not syntactically equal, have the same behavior with respect to failure, termination, and effect-preservation.

□

Received 13th April 2007

Acta Cybernetica 18 (2008) 777–782.

On Monogenic Nondeterministic Automata*

Csanád Imreh† and Masami Ito‡

Abstract

A finite automaton is said to be directable if it has an input word, a directing word, which takes it from every state into the same state. For nondeterministic (n.d.) automata, directability can be generalized in several ways, three such notions, D1-, D2-, and D3-directability, are used. In this paper, we consider monogenic n.d. automata, and for each $i = 1, 2, 3$, we present sharp bounds for the maximal lengths of the shortest Di-directing words.

1 Introduction

An input word w is called a *directing* (or *synchronizing) word* of an automaton $\mathcal{A}$ if it takes $\mathcal{A}$ from every state to the same state. Directable automata have been studied extensively. In the famous paper of Cerńy [4] it was conjectured that the shortest directing word of an n-state directable automaton has length at most $(n-1)^2$. The best known upper bound on the length of the shortest directing words is $(n^3-n)/6$ (see [5] and [7]). The same problem was investigated for several subclasses of automata. We do not list here these results but we just mention the most recent paper on the subclass of monotonic automata [1]. Further results on subclasses are mentioned in that paper, and in the papers listed in its references.

Directable n.d. automata have been obtained a fewer interest. Directability to n.d. automata can be extended in several meaningful ways. The following three nonequivalent definitions are introduced and studied in [11]. An input word w of an n.d. automaton $\mathcal{A}$ is said to be

(1) D1-*directing* if it takes $\mathcal{A}$ from every state to the same singleton set,

(2) D2-*directing* if it takes $\mathcal{A}$ from every state to the same fixed set A', where $\emptyset \subseteq A' \subseteq A$,

(3) D3-*directing* if there is a state c such that $c \in aw$, for every $a \in A$.

*This work has been supported by a collaboration between the Hungarian Academy of Science and the Japan Society for the Promotion of Science.

†Department of Informatics, University of Szeged, Árpád tér 2, H-6720 Szeged, Hungary

‡Dept. of Mathematics, Faculty of Science, Kyoto Sangyo University, Kyoto 603-8555, Japan

The D1-directability of complete n.d. automata was investigated by Burkhard [2]. He gave a sharp exponential bound for the lengths of minimum-length D1-directing words of complete n.d. automata. Goralčik *et al.* [6] studied D1- and D3-directability and they proved that neither for D1- nor for D3-directing words, the bound can be polynomial for n.d. automata. These bounds are improved in [13], one can find an overview of the the results on directing words of n.d. automata in the book [12].

Carpi [3] considered a particular class of n.d. automata, the class of unambigous n.d. automata, and presented $O(n^3)$ bounds for the lengths of their shortest D1-directing words. Trapped n.d. automata are investigated in [8], monotonic n.d. automata are investigated in [9], and commutative n.d. automata are investigated in [10].

In this work, we study the class of *monogenic n.d. automata*, the subclass where the alphabet contains only one symbol. This class is a subclass of the commutative n.d. automata. Shortest directing words of the monogenic and commutative automata are investigated in [14] and [15]. We prove tight bounds for monogenic n.d. automata on the lengths of shortest directing words of each type.

2 Notions and notations

Let X denote a finite nonempty alphabet. The set of all finite words over X is denoted by X^* and λ denotes the empty word. The length of a word $w \in X^*$ is denoted by $|w|$.

By a *nondeterministic (n.d.) automaton* we mean a system $\mathcal{A} = (A, X)$, where A is a nonempty finite set of *states*, X is the *input alphabet*, and each input symbol $x \in X$ is realized as a binary relation $x^{\mathcal{A}} (\subseteq A \times A)$. For any $a \in A$ and $x \in X$, let

$$ax^{\mathcal{A}} = \{b : b \in A \text{ and } (a, b) \in x^{\mathcal{A}}\}.$$

Moreover, for every $B \subseteq A$, we denote by $Bx^{\mathcal{A}}$ the set $\cup\{ax^{\mathcal{A}} : a \in B\}$. Now, for any word $w \in X^*$ and $B \subseteq A$, $Bw^{\mathcal{A}}$ can be defined inductively as follows:

(1) $B\lambda^{\mathcal{A}} = B$,

(2) $Bw^{\mathcal{A}} = (Bp^{\mathcal{A}})x^{\mathcal{A}}$ for $w = px$, where $p \in X^*$ and $x \in X$.

If $w = x_1 \dots x_m$ and $a \in A$, then let $aw^{\mathcal{A}} = \{a\}w^{\mathcal{A}}$. This yields that $w^{\mathcal{A}} = x_1^{\mathcal{A}} \dots x_m^{\mathcal{A}}$. If there is no danger of confusion, then we write simply aw and Bw for $aw^{\mathcal{A}}$ and $Bw^{\mathcal{A}}$, respectively.

Following [11], we define the directability of n.d. automata as follows. Let $\mathcal{A} = (A, X)$ be an n.d. automaton. For any word $w \in X^*$, let us consider the following conditions:

(D1) $(\exists c \in A)(\forall a \in A)(aw = \{c\})$,

(D2) $(\forall a, b \in A)(aw = bw)$,

(D3) $(\exists c \in A)(\forall a \in A)(c \in aw)$.

For any $i = 1, 2, 3$, if w satisfies Di, then w is called a Di-*directing word* of $\mathcal{A}$ and in this case $\mathcal{A}$ is said to be Di-*directable*. Let us denote by $\mathrm{D}_i(\mathcal{A})$ the set of Di-directing words of $\mathcal{A}$. Moreover, let $\mathbf{Dir}(i)$ denote the classes of Di-directable n.d. automata. Now, we can define the following functions. For any $i = 1, 2, 3$ and $\mathcal{A} = (A, X) \in \mathbf{Dir}(i)$, let

$$\mathrm{d}_i(\mathcal{A}) = \min\{|w| : w \in \mathrm{D}_i(\mathcal{A})\},$$

$$\mathrm{d}_i(n) = \max\{\mathrm{d}_i(\mathcal{A}) : \mathcal{A} \in \mathbf{Dir}(i) \ \&\ |A| = n\}.$$

The functions $\mathrm{d}_i(n)$, $i = 1, 2, 3$, are studied in [11] and [13], where lower and upper bounds depending on n are presented for them. Similar functions can be defined for any class of n.d. automata. For a class $\mathbf{K}$ of n.d. automata, let

$$\mathrm{d}_i^{\mathbf{K}}(n) = \max\{\mathrm{d}_i(\mathcal{A}) : \mathcal{A} \in \mathbf{Dir}(i) \cap \mathbf{K} \ \&\ |A| = n\}.$$

3 Monogenic n.d. automata

In what follows, we study the case when the considered class is $\mathbf{MG}$, the class of monogenic n.d. automata. For the class $\mathbf{C}$ of commutative automata it is shown in [10] that $\mathrm{d}_1^{\mathbf{C}}(n) = (n-1)$. Since every monogenic n.d. automaton is commutative we obtain that $\mathrm{d}_1^{\mathbf{MG}}(n) \leq (n-1)$. Moreover, the n.d. automaton which proves in [10] that $\mathrm{d}_1^{\mathbf{C}}(n) \geq (n-1)$ is a monogenic one and thus we obtain immediately the following corollary.

Corollary 1. *For any* $n \geq 1$, $\mathrm{d}_1^{\mathbf{MG}}(n) = (n-1)$.

For the $D2$-directable monogenic n.d. automaton we have the following result.

Theorem 1. *For any* $n \geq 2$, $\mathrm{d}_2^{\mathbf{MG}}(n) = (n-1)^2 + 1$.

Proof. To prove that $\mathrm{d}_2^{\mathbf{MG}}(n) \geq (n-1)^2+1$ we can use the same n.d. automaton which was used in [10]. For the sake of completeness we recall the definition of the automaton here. The set of states is $S = \{1, \ldots, n\}$, there is one letter in the alphabet denoted by x, and it is defined as follows: $1x = \{1, 2\}$, $ix = \{i+1\}$ for $1 < i < n$, and $nx = \{1\}$. It is easy to see that the shortest $D2$-directing word of this n.d. automaton has length $(n-1)^2 + 1$.

Now we prove that $\mathrm{d}_2^{\mathbf{MG}}(n) \leq (n-1)^2 + 1$. We prove it by induction on n. If $n = 2$ then the statement is obviously valid. Let $n \geq 2$ and suppose that the inequality is valid for each $i < n$. Consider an arbitrary monogenic $D2$-directable n.d. automaton with n states. Let denote the set of states by $S = \{1, \ldots, n\}$ and the letter in the alphabet by x. Let m be the length of the shortest $D2$-directing word. This means that $ix^m = jx^m$ for each $i, j \in S$.

Suppose first that $Sx \subset S$. Then consider the n.d. automaton (Sx, x). This is a $D2$-directable monogenic n.d. automaton with less than n states. Thus its shortest $D2$-directing word has length at most $(n-2)^2 + 1$. Therefore, the original n.d.

automaton has a $D2$-directing word with length at most $(n-2)^2+2 \leq (n-1)^2+1$ and this proves the statement in this case.

Therefore, we can suppose that $Sx = S$. This yields that $Sx^k = S$ for each k. Thus $ix^m = Sx^m = S$ for each $i \in S$. Let $i \in S$ be arbitrary and consider the sequence of sets $\{i\}, ix, ix^2 \ldots$. If $ix^k = S$, then $ix^l = S$ for each $l \geq k$. Now suppose that $ix^k = ix^l$, $k < l$. Then the sequence of the sets becomes a periodic sequence from ix^k with the period $k - l$, and thus this case is only possible if $ix^k = ix^l = S$.

Let p be the smallest positive value with the property $i \in ix^p$. Since $i \in ix^m$ is valid, such p exists. Then we have $i \in ix^p$. Furthermore, $\{i\} \neq ix^p$ therefore, $|ix^p| \geq 2$. On the other hand by $i \in ix^p$ it also holds that $ix^{qp} \subseteq ix^{(q+1)p}$ and this yields that if $ix^{qp} \neq S$ then $|ix^{(q+1)p}| > |ix^{qp}|$. Thus we obtain that $ix^{(n-1)p} = S$.

Now consider the following sets. Let $H_j = \cup_{k=1}^{j} ix^k$. Then $H_j \subseteq H_{j+1}$ for each j. Furthermore, if $H_j = H_{j+1}$ for some j then $H_j = H_k$ for each $k \geq j$, therefore, this is only possible in the case when $H_j = S$.

Let r be the smallest positive value with the property $|ix^r| \geq 2$. Consider the following two cases.

Case I. Suppose that $ix^r \cap H_{r-1} = \emptyset$. In this case $|H_r| \geq |H_{r-1}| + 2$, thus we obtain that $H_{n-1} = S$. This yields that $p \leq n-1$ and it follows that $(n-1)p < (n-1)^2 + 1$.

Case II. Suppose that there exists j such that $j \in ix^r \cap H_{r-1}$. Then there exists $s < r$ such that $ix^s = \{j\}$. Then for each $t \geq 0$ we have $ix^{s+t(r-s)} \subseteq ix^{s+(t+1)(r-s)}$. Since these sets can be equal only in the case when they are equal to S we obtain that $ix^{s+(n-1)(r-s)} = S$. On the other hand $r \leq n$ and $s \geq 1$ thus we obtain that $s + (n-1)(r-s) \leq (n-1)^2 + 1$.

For the $D3$-directable monogenic n.d. automaton we have the following result.

Theorem 2. *For any $n \geq 1$, $\mathrm{d}_3^{\mathbf{MG}}(n) = n^2 - 3n + 3$.*

To prove that $\mathrm{d}_3^{\mathbf{MG}}(n) \geq n^2 - 3n + 3$ we can use the same n.d. automaton which was used in the $D2$-directable case. In [12] it is shown that the shortest $D3$-directing word of this n.d. automaton has length $n^2 - 3n + 3$.

Now we prove that $\mathrm{d}_3^{\mathbf{MG}}(n) \leq n^2 - 3n + 3$. Consider an arbitrary monogenic $D3$-directable n.d. automaton with n states. Let denote the set of states by $S = \{1, \ldots, n\}$ and the letter in the alphabet by x. Let m be the length of the shortest $D3$-directing word. Then there exists a state i with the property $i \in jx^m$ for each $j \in S$.

Define the following n.d. automaton. Let $\mathcal{B} = (S, y)$, where the transition y is defined by the rule $jy = \{k \in S \ : \ j \in kx\}$. Then we obtain by induction that $jy^p = \{k \in S \mid j \in kx^p\}$ Therefore, $iy^m = S$. Moreover, it holds that $jy^p \neq S$ for all $p < m$ and $j \in S$ since otherwise we would obtain a shorter $D3$-directing word than a^m.

Now we can use a similar technique to finish the proof as we did in the case of D2-directability. Consider the sequence of sets $\{i\}, iy, iy^2 \ldots, iy^m$. This sequence contains different sets and $iy^m = S$. Let us observe that $|iy| \geq 2$, otherwise by $\{iy\}y^{m-1} = S$ we would obtain a contradiction.

Let p be the smallest positive value with the property $i \in iy^p$. Then we have $iy \subseteq iy^{p+1}$. Furthermore, $\{iy\} \neq iy^{p+1}$ and therefore, $|iy^{p+1}| \geq 3$. On the other hand by $iy \subseteq iy^{p+1}$ it also holds that $iy^{qp+1} \subseteq iy^{(q+1)p+1}$ and this yields that if $iy^{qp+1} \neq S$ then $|iy^{(q+1)p+1}| > |iy^{qp+1}|$. Thus we obtain that $iy^{(n-2)p+1} = S$.

If $i \in iy$ then $p = 1$. Otherwise consider the sets $H_j = \cup_{k=1}^{j} iy^k$. Then $H_j \subset H_{j+1}$ for each j, if $H_j \neq S$. Since $|H_1| \geq 2$ we obtain that $H_{n-1} = S$. This yields that $p \leq n-1$.

Therefore, we obtained that $ix^{(n-2)(n-1)+1} = S$ which proves that $m \leq (n-2)(n-1)+1 = n^2 - 3n + 3$.

References

[1] D. S. Ananichev, M. V. Volkov, Synchronizing generalized monotonic automata. Theoret. Comput. Sci. 330 (2005), no. 1, 3–13.

[2] H. V. Burkhard, Zum Längenproblem homogener Experimente an determienten und nicht-deterministischen Automata, *Electronische Informationsverarbbeitung und Kybernetik, EIK* **12** (1976), 301-306.

[3] A. Carpi, On synchronizing unambigous automata, *Theoretical Computer Science* **60** (1988), 285-296.

[4] J. Černý, Poznáamka k homogénym experimentom ss konecinými automatami, *Mat.-fiz. cas. SAV* **14** (1964), 208–215.

[5] P. Frankl, An extremal problem for two families of sets. European J. Combin. 3 (1982), no. 2, 125–127.

[6] P. Goralčik, Z. Hedrlin, V. Koubek, J. Ryšlinková, A game of composing binary relations, *R.A.I.O. Informatique théorique/Theoretical Informatics* **16** (1982), 365–369.

[7] J.-E. Pin, On two combinatorial problems arising from automata theory, Annals of Discrete Mathematics 17 (1983), 535-548.

[8] B. Imreh, Cs. Imreh, M. Ito, On directable nondeterministic trapped automata *Acta Cybernetica*, **16**, 2003, 37–45.

[9] B. Imreh, Cs. Imreh, M. Ito, On Monotonic Directable Nondeterministic Automata, *Journal of Automata, Languages and Combinatorics*, **8**, 2003, 539–547.

[10] B. Imreh, M. Ito, M. Steinby, On Commutative Directable Nondeterministic Automata. Grammars and Automata for String Processing 2003: 141-150

[11] B. Imreh, M. Steinby, Directable nondeterministic automata, *Acta Cybernetica* **14** (1999), 105-115.

[12] M. Ito, *Algebraic theory of automata and languages.* World Scientific Publishing Co., Inc., River Edge, NJ, 2004.

[13] M. Ito, K. Shikishima-Tsuji, Some results on directable automata. *Theory is forever, LNCS 3113*, Springer, Berlin, 2004, 125–133.

[14] I. Rystsov, Exact linear bound for the length of reset words in commutative automata. *Publ. Math. Debrecen* 48 (1996), no. 3-4, 405–409.

[15] I. Rystsov, Reset words for commutative and solvable automata, *Theoretical Computer Science* **172** (1997), 273-279.

Received 28th March 2007

Acta Cybernetica 18 (2008) 783–793.

Two Power-Decreasing Derivation Restrictions in Generalized Scattered Context Grammars*

Tomáš Masopust† Alexander Meduna† and Jiří Šimáček†

Abstract

The present paper introduces and discusses generalized scattered context grammars that are based upon sequences of productions whose left-hand sides are formed by nonterminal strings, not just single nonterminals. It places two restrictions on the derivations in these grammars. More specifically, let k be a positive integer. The first restriction requires that all rewritten symbols occur within the first k symbols of the first continuous block of nonterminals in the sentential form during every derivation step. The other restriction defines derivations over sentential forms containing no more than k occurrences of nonterminals. As its main result, the paper demonstrates that both restrictions decrease the generative power of these grammars to the power of context-free grammars.

Keywords: scattered context grammar; grammatical generalization; derivation restriction; generative power.

1 Introduction

Scattered context grammars are based upon finite sets of sequences of context-free productions having a single nonterminal on the left-hand side of every production (see [5]). According to a sequence of n context-free productions, these grammars simultaneously rewrites n nonterminals in the current sentential form according to the n productions in the order corresponding to the appearance of these productions in the sequence. It is well-known that they characterize the family of recursively enumerable languages (see [8]).

In this paper, we generalize these grammars so that the left-hand side of every production may consist of a string of several nonterminals rather than a single nonterminal. Specifically, we discuss two derivation restrictions in scattered context grammars generalized in this way. To explain these restrictions, let k be a constant. The first restriction requires that all simultaneously rewritten symbols

*This work was supported by the Czech Ministry of Education under the Research Plan No. MSM 0021630528 and the Czech Grant Agency project No. 201/07/0005.

†Faculty of Information Technology, Brno University of Technology, Božetěchova 2, Brno 61266, Czech Republic, E-mail: {masopust,meduna}@fit.vutbr.cz, xsimac00@stud.fit.vutbr.cz

occur within the first k symbols of the first continuous block of nonterminals in the current sentential form during every derivation step. The other restriction defines the grammatical derivations over sentential forms containing no more than k occurrences of nonterminals. As the main result, this paper demonstrates that both restrictions decrease the generative power of generalized scattered context grammars to the generative power of context-free grammars. As ordinary scattered context grammars represent special cases of their generalized versions, they also characterize only the family of context-free languages if they are restricted in this way.

This result concerning the derivation restrictions is of some interest when compared to analogical restrictions in terms of other grammars working in a context-sensitive way. Over its history, formal language theory has studied many restrictions placed on the way grammars derive sentential forms and on the forms of productions. In [6], Matthews studied derivations of grammars in the strictly leftmost (rightmost) way—that is, rewritten symbols are preceded (succeeded) only by terminals in the sentential form during the derivation. Later, in [7], he combined both approaches—leftmost and rightmost derivations—so that any sentential form during the derivation is of the form xWy, where x and y are terminal strings, W is a nonterminal string, and a production is applicable only to a leftmost or rightmost substring of W. In both cases, these restrictions result into decreasing the generative power of type-0 grammars to the power of context-free grammars.

Whereas Matthews studied restrictions placed on the forms of derivations, other authors studied the forms of productions. In [2], Book proved that if the left-hand side of any non-context-free production contains besides exactly one nonterminal only terminals, then the generative power of type-0 grammars decreases to the power of context-free grammars. He also proved that if the left-hand side of any non-context-free production has as its left context a terminal string and the left context is at least as long as the right context, then the generative power of type-0 grammars decreases to the power of context-free grammars, too. In [4], Ginsburg and Greibach proved that if the left-hand side of any production is a nonterminal string and the right-hand side contains at least one terminal, then the generated language is context-free. Finally, in [1], Baker proved a stronger result. This result says that if any left-hand side of a production either has, besides terminals, only one nonterminal, or there is a terminal substring, β, on the right-hand side of the production such that the length of β is greater than the length of any terminal substring of the left-hand side of the production, then the generative power of type-0 grammars decreases to the power of context-free grammars. For more details, see page 198 in [9] and the literature cited there.

2 Preliminaries

In this paper, we assume that the reader is familiar with formal language theory (see [10]). For a set Q, $|Q|$ denotes the cardinality of Q. For an alphabet (finite nonempty set) V, V^* represents the free monoid generated by V. The identity of

V^* is denoted by ε. Set $V^+ = V^* - \{\varepsilon\}$. For $w \in V^*$, $|w|$ and w^R denote the length and the mirror image of w, respectively, and $sub(w)$ denotes the set of all substrings of w. For $W \subseteq V$, $occur(w, W)$ denotes the number of occurrences of symbols from W in w.

A *pushdown automaton* is a septuple $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$, where Q is a finite set of states, Σ is an input alphabet, $q_0 \in Q$ is the initial state, Γ is a pushdown alphabet, δ is a finite set of rules of the form $Zqa \to \gamma p$, where $p, q \in Q$, $Z \in \Gamma \cup \{\varepsilon\}$, $a \in \Sigma \cup \{\varepsilon\}$, $\gamma \in \Gamma^*$, F is a set of final states, and Z_0 is the initial pushdown symbol. Let ψ denote a bijection from δ to Ψ (Ψ is an alphabet of rule labels). We write $r.Zqa \to \gamma p$ instead of $\psi(Zqa \to \gamma p) = r$.

A configuration of M is any word from $\Gamma^* Q \Sigma^*$. For any configuration $xAqay$, where $x \in \Gamma^*$, $y \in \Sigma^*$, $q \in Q$, and any $r.Aqa \to \gamma p \in \delta$, M makes a move from $xAqay$ to $x\gamma py$ according to r, written as $xAqay \Rightarrow x\gamma py\,[r]$, or, simply, $xAqay \Rightarrow x\gamma py$. If $x, y \in \Gamma^* Q \Sigma^*$ and $m > 0$, then $x \Rightarrow^m y$ if and only if there exists a sequence $x_0 \Rightarrow x_1 \Rightarrow \cdots \Rightarrow x_m$, where $x_0 = x$ and $x_m = y$. Then, we say $x \Rightarrow^+ y$ if and only if there exists $m > 0$ such that $x \Rightarrow^m y$, and $x \Rightarrow^* y$ if and only if $x = y$ or $x \Rightarrow^+ y$. The language of M is defined as $\mathcal{L}(M) = \{w \in \Sigma^* : Z_0 q_0 w \Rightarrow^* f, f \in F\}$.

A *phrase-structure grammar* or a *grammar* is a quadruple $G = (V, T, P, S)$, where V is a total alphabet, $T \subseteq V$ is an alphabet of terminals, $S \in V - T$ is the start symbol, and P is a finite relation over V^*. Set $N = V - T$. Instead of $(u, v) \in P$, we write $u \to v \in P$ throughout. We call $u \to v$ a production; accordingly, P is G's set of productions. If $u \to v \in P$, $x, y \in V^*$, then G makes a derivation step from xuy to xvy, symbolically written as $xuy \Rightarrow xvy$. If $x, y \in V^*$ and $m > 0$, then $x \Rightarrow^m y$ if and only if there exists a sequence $x_0 \Rightarrow x_1 \Rightarrow \cdots \Rightarrow x_m$, where $x_0 = x$ and $x_m = y$. We write $x \Rightarrow^+ y$ if and only if there exists $m > 0$ such that $x \Rightarrow^m y$, and $x \Rightarrow^* y$ if and only if $x = y$ or $x \Rightarrow^+ y$. The language of G is defined as $\mathcal{L}(G) = \{w \in T^* : S \Rightarrow^* w\}$.

3 Definitions

This section defines a new notion of generalized scattered context grammars. In addition, it formalizes two derivation restrictions studied in this paper.

A *generalized scattered context grammar*, a **SCG** for short, is a quadruple $G = (V, T, P, S)$, where V is a total alphabet, $T \subseteq V$ is an alphabet of terminals, $S \in N$ ($N = V - T$) is the start symbol, and P is a finite set of productions such that each production p has the form $(\alpha_1, \ldots, \alpha_n) \to (\beta_1, \ldots, \beta_n)$, for some $n \geq 1$, where $\alpha_i \in N^+$, $\beta_i \in V^*$, for all $1 \leq i \leq n$. If each production p of the above form satisfies $|\alpha_i| = 1$, for all $1 \leq i \leq n$, then G is an ordinary *scattered context grammar*. Set $\pi(p) = n$. If $\pi(p) \geq 2$, then p is said to be a *context-sensitive* production. If $\pi(p) = 1$, then p is said to be *context-free*. If $(\alpha_1, \ldots, \alpha_n) \to (\beta_1, \ldots, \beta_n) \in P$, $u = x_0 \alpha_1 x_1 \ldots \alpha_n x_n$, and $v = x_0 \beta_1 x_1 \ldots \beta_n x_n$, where $x_i \in V^*$, $1 \leq i \leq n$, then $u \Rightarrow v\ [(\alpha_1, \ldots, \alpha_n) \to (\beta_1, \ldots, \beta_n)]$ in G or, simply, $u \Rightarrow v$. Let $\Rightarrow^+$ and $\Rightarrow^*$ denote the transitive and the reflexive and transitive closure of $\Rightarrow$, respectively. The language of G is defined as $\mathcal{L}(G) = \{w \in T^* : S \Rightarrow^* w\}$.

For an alphabet $T = \{a_1, \ldots, a_n\}$, there is an *extended Post correspondence problem*, E, defined as

$$E = (\{(u_1, v_1), \ldots, (u_r, v_r)\}, (z_{a_1}, \ldots, z_{a_n}))\,,$$

where $u_i, v_i, z_{a_j} \in \{0,1\}^*$, for each $1 \leq i \leq r$, $1 \leq j \leq n$. The language represented by E is the set

$$\mathcal{L}(E) = \{b_1 \ldots b_k \in T^* : \text{ exists } s_1, \ldots, s_l \in \{1, \ldots, r\}, l \geq 1, \\ v_{s_1} \ldots v_{s_l} = u_{s_1} \ldots u_{s_l} z_{b_1} \ldots z_{b_k} \text{ for some } k \geq 0\}.$$

It is well known that for each recursively enumerable language, L, there is an extended Post correspondence problem, E, such that $\mathcal{L}(E) = L$ (see Theorem 1 in [3]).

Next, we define two derivation restrictions discussed in this paper.

Let $k \geq 1$. If there is $(\alpha_1, \ldots, \alpha_n) \to (\beta_1, \ldots, \beta_n) \in P$, $u = x_0\alpha_1 x_1 \ldots \alpha_n x_n$, and $v = x_0\beta_1 x_1 \ldots \beta_n x_n$, where

1. $x_0 \in T^*N^*$,
2. $x_i \in N^*$, for all $0 < i < n$,
3. $x_n \in V^*$, and
4. $\text{occur}(x_0\alpha_1 x_1 \ldots \alpha_n, N) \leq k$,

then $u \;{}_k\!\Longrightarrow v$ $[r]$ in G or, simply, $u \;{}_k\!\Longrightarrow v$. Let ${}_k\!\Longrightarrow^n$ denote the n-fold product of ${}_k\!\Longrightarrow$, where $n \geq 0$. Furthermore, let ${}_k\!\Longrightarrow^*$ denote the reflexive and transitive closure of ${}_k\!\Longrightarrow$. Set ${}_{k-left}\mathcal{L}(G) = \{w \in T^* : S \;{}_k\!\Longrightarrow^* w\}$.

Let $m, h \geq 1$. $W(m)$ denotes the set of all strings $x \in V^*$ satisfying 1 given next. $W(m,h)$ denotes the set of all strings $x \in V^*$ satisfying 1 and 2 given next.

1. $x \in (T^*N^*)^m T^*$;
2. $(y \in \text{sub}(x)$ and $|y| > h)$ implies $\text{alph}(y) \cap T \neq \emptyset$.

If there is $(\alpha_1, \ldots, \alpha_n) \to (\beta_1, \ldots, \beta_n) \in P$, $u = x_0\alpha_1 x_1 \ldots \alpha_n x_n$, and $v = x_0\beta_1 x_1 \ldots \beta_n x_n$, where

1. $x_0 \in V^*$,
2. $x_i \in N^*$, for all $0 < i < n$, and
3. $x_n \in V^*$,

then $u \Longrightarrow v$ $[r]$ in G or, simply, $u \Longrightarrow v$. Let $\Longrightarrow^n$ denote n-fold product of $\Longrightarrow$, where $n \geq 0$. Furthermore, let $\Longrightarrow^*$ denote the reflexive and transitive closure of $\Longrightarrow$.

Let $u, v \in V^*$, and $u \Longrightarrow v$.

$$u \;{}^h_m\!\Longrightarrow v$$

if and only if $u, v \in W(m,h)$, and

$$u \;{}_m\!\Longrightarrow v$$

if and only if $u, v \in W(m)$. Set ${}_{nonter}\mathcal{L}(G,m,h) = \{w \in T^* : S \;{}^h_m\!\Longrightarrow^* w\}$ and ${}_{nonter}\mathcal{L}(G,m) = \{w \in T^* : S \;{}_m\!\Longrightarrow^* w\}$.

3.1 Language Families

Let **SCGs** denote the family of generalized scattered context grammars. Define these language families:

$$\begin{aligned} {}_{nonter}SC(m,h) &= \{L : L = {}_{nonter}\mathcal{L}(G,m,h), G \in \mathbf{SCGs}\} \text{ for all } m,h \geq 1 \\ {}_{nonter}SC(m) &= \{L : L = {}_{nonter}\mathcal{L}(G,m), G \in \mathbf{SCGs}\} \text{ for all } m \geq 1 \\ {}_{k-left}SC &= \{L : L = {}_{k-left}\mathcal{L}(G), G \in \mathbf{SCGs}\} \text{ for all } k \geq 0 \end{aligned}$$

Let **CF**, **CS**, and **RE** denote the families of context-free, context-sensitive, and recursively enumerable languages, respectively. For all $k \geq 0$, ${}_kCF$ denote the family of languages generated by context-free grammars of index k.

4 Results

This section presents the main results of this paper. First, it demonstrates that, for every $k \geq 1$, $CF = {}_{k-left}SC$, then that $RE = {}_{nonter}SC(1)$, and, finally, that for every $m,h \geq 1$, ${}_mCF = {}_{nonter}SC(m,h)$.

Theorem 1. *Let k be a positive integer. Then, $CF = {}_{k-left}SC$.*

Proof. Let $G = (V,T,P,S)$ be a generalized scattered context grammar. Consider the following pushdown automaton

$$M = (\{q,r,f\} \cup \{[\gamma,s] : \gamma \in N^*, |\gamma| \leq k, s \in \{q,r\}\}, T, V \cup \{Z\}, \delta, [S,q], Z, \{f\}),$$

where $Z \notin V$, and δ contains rules of the following forms:

1. $[\beta_0 A_1 \beta_1 \dots A_n \beta_n, q] \to (\beta_0 \alpha_1 \beta_1 \dots \alpha_n \beta_n)^R [\varepsilon, r]$ if $(A_1,\dots,A_n) \to (\alpha_1,\dots,\alpha_n) \in P$; $\beta_i \in N^*$, $0 \leq i \leq n$;
2. $A[A_1 \dots A_n, r] \to [A_1 \dots A_n A, r]$ if $n < k$, $A \in N$;
3. $[A_1 \dots A_k, r] \to [A_1 \dots A_k, q]$;
4. $a[A_1 \dots A_n, r] \to a[A_1 \dots A_n, q]$ if $n < k$, $a \in T$;
5. $Z[A_1 \dots A_n, r] \to Z[A_1 \dots A_n, q]$ if $n < k$;
6. $a[\varepsilon, r]a \to [\varepsilon, r]$ if $a \in T$;
7. $Z[\varepsilon, r] \to f$.

We prove that $\mathcal{L}(M) = {}_{k-left}\mathcal{L}(G)$.

($\subseteq$:) By induction on the number of rules constructed in 1 used in a sequence of moves, we prove the following claim.

Claim 1. *If $Z\alpha^R[\beta_0 A_1 \beta_1 \dots A_n \beta_n, q]w \Rightarrow^* f$, then $\beta_0 A_1 \beta_1 \dots A_n \beta_n \alpha \; {}_k\!\Leftarrow^* w$.*

Proof. Basis: Only one rule constructed in 1 is used. Then,

$$Z\alpha^R[\beta_0 A_1 \beta_1 \dots A_n \beta_n, q]uw \Rightarrow Z(\beta_0 \alpha_1 \beta_1 \dots \alpha_n \beta_n \alpha)^R [\varepsilon, r]uw \Rightarrow^* f,$$

where $(A_1,\ldots,A_n) \to (\alpha_1,\ldots,\alpha_n) \in P$, $n \leq k$, and $\beta_0\alpha_1\beta_1\ldots\alpha_n\beta_n\alpha \in T^*$. Therefore, $\beta_0 = \cdots = \beta_n = \varepsilon$, and $\alpha_1\ldots\alpha_n\alpha = uw$. Then,

$$A_1\ldots A_n w \ {}_k\!\Leftrightarrow uw.$$

Induction hypothesis: Suppose that the claim holds for all sequences of moves containing no more than i rules constructed in 1.

Induction step: Consider a sequence of moves containing $i+1$ rules constructed in 1. Then,

$$\begin{array}{lll} & Z\alpha^R[\beta_0A_1\beta_1\ldots A_l\beta_l,q]w & \\ \Rightarrow & Z\alpha^R(\beta_0\alpha_1\beta_1\ldots\alpha_l\beta_l)^R[\varepsilon,r]w & \text{(by a rule constructed in 1)} \\ \Rightarrow^* & Z\alpha'[\varepsilon,r]w' & \text{(by rule constructed in 6)} \\ \Rightarrow^* & Z\alpha''[\beta'_0B_1\beta'_1\ldots B_m\beta'_m,r]w' & \text{(by rule constructed in 2)} \\ \Rightarrow & Z\alpha''[\beta'_0B_1\beta'_1\ldots B_m\beta'_m,q]w' & \text{(by a rule constructed in 3, 4, or 5)} \\ \Rightarrow^* & f & \end{array}$$

where $\alpha' \in V^*N \cup \{\varepsilon\}$, $v \in T^*$, $\alpha'v^R = \alpha^R(\beta_0\alpha_1\beta_1\ldots\alpha_l\beta_l)^R$, and $vw' = w$. Then, by the production $(A_1,\ldots,A_l) \to (\alpha_1,\ldots,\alpha_l)$,

$$\beta_0A_1\beta_1\ldots A_l\beta_l\alpha \ {}_k\!\Leftrightarrow \beta_0\alpha_1\beta_1\ldots\alpha_l\beta_l\alpha,$$

where $|\beta_0A_1\beta_1\ldots A_l\beta_l| \leq k$,

$$\beta_0\alpha_1\beta_1\ldots\alpha_l\beta_l\alpha = v(\alpha')^R = v\beta'_0B_1\beta'_1\ldots B_m\beta'_m(\alpha'')^R,$$

and, by the induction hypothesis,

$$v\beta'_0B_1\beta'_1\ldots B_m\beta'_m(\alpha'')^R \ {}_k\!\Leftrightarrow^* vw'.$$

Hence, the inclusion holds. △

($\supseteq$:) First, we prove the following claim.

Claim 2. *If* $\beta \ {}_k\!\Leftrightarrow^* w$, *where* $\beta \in NV^*$, *then* $Z\beta^R[\varepsilon,r]w \Rightarrow^* f$.

Proof. By induction on the length of derivations.

Basis: Let $A_1\ldots A_nw \ {}_k\!\Leftrightarrow \alpha_1\ldots\alpha_nw$ $(\alpha_1\ldots\alpha_n = \alpha)$, where $\alpha w \in {}_{k-left}\mathcal{L}(G)$, and $(A_1,\ldots,A_n) \to (\alpha_1,\ldots,\alpha_n) \in P$, $1 \leq n \leq k$. M simulates this derivation step as follows.

$$\begin{array}{lll} & Zw^RA_n\ldots A_1[\varepsilon,r]\alpha w & \\ \Rightarrow^n & Zw^R[A_1\ldots A_n,r]\alpha w & \text{(by rule constructed in 2)} \\ \Rightarrow & Zw^R[A_1\ldots A_n,q]\alpha w & \text{(by a rule constructed in 4 or 5)} \\ \Rightarrow & Zw^R\alpha^R[\varepsilon,r]\alpha w & \text{(by a rule constructed in 1)} \\ \Rightarrow^{|\alpha w|} & Z[\varepsilon,r] & \text{(by rule constructed in 6)} \\ \Rightarrow & f & \text{(by the rule constructed in 7)} \end{array}$$

Induction hypothesis: Suppose that the claim holds for all derivations of length i or less.

Induction step: Consider a derivation of length $i+1$. Let

$$\beta_0 B_1 \beta_1 \dots B_l \beta_l \gamma \ {}_k\Leftrightarrow\ \beta_0 \alpha_1 \beta_1 \dots \alpha_l \beta_l \gamma \ {}_k\Leftrightarrow^i \varphi w,$$

where $\varphi w \in {}_{k-left}\mathcal{L}(G)$, $\beta_0 B_1 \beta_1 \dots B_l \beta_l \in N^+$, and either $|\beta_0 B_1 \beta_1 \dots B_l \beta_l| = k$, or $|\beta_0 B_1 \dots B_l \beta_l| < k$, $\beta_0 \alpha_1 \beta_1 \dots \alpha_l \beta_l \gamma = \varphi\psi$, where $\varphi \in T^*$, $\psi \in NV^* \cup \{\varepsilon\}$, and $\gamma \in TV^* \cup \{\varepsilon\}$. Then,

	$Z(\beta_0 B_1 \beta_1 \dots B_l \beta_l \gamma)^R[\varepsilon, r]\varphi w$	
$\Rightarrow^*$	$Z\gamma^R[\beta_0 B_1 \beta_1 \dots B_l \beta_l, r]\varphi w$	(by rule constructed in 2)
$\Rightarrow$	$Z\gamma^R[\beta_0 B_1 \beta_1 \dots B_l \beta_l, q]\varphi w$	(by a rule constructed in 3 or 4)
$\Rightarrow$	$Z(\varphi\psi\gamma)^R[\varepsilon, r]\varphi w$	(by a rule constructed in 1)
$\Rightarrow^*$	$Z(\psi\gamma)^R[\varepsilon, r]w$	(by a rule constructed in 6)
$\Rightarrow^*$	f	(by the induction hypothesis)

Hence, the claim holds. △

Now, if $S \Rightarrow u\alpha \Rightarrow^* uw$, where $u \in T^*$ and $\alpha \in NV^*$, then $Z[S, q]uw \Rightarrow Z(u\alpha)^R[\varepsilon, r]uw \Rightarrow^* Z\alpha^R[\varepsilon, r]w \Rightarrow^* f$, by rules constructed in 1 and 6 and the previous claim. For $\alpha = \varepsilon$, $Z[S, q]u \Rightarrow Zu^R[\varepsilon, r]u \Rightarrow^* f$. Hence, the other inclusion holds. □

Theorem 2. $RE = {}_{nonter}SC(1)$.

Proof. Let $L \subseteq \{a_1, \dots, a_n\}^*$ be a recursively enumerable language. There is an extended Post correspondence problem,

$$E = (\{(u_1, v_1), \dots, (u_r, v_r)\}, (z_{a_1}, \dots, z_{a_n})),$$

where $u_i, v_i, z_{a_j} \in \{0,1\}^*$, for each $1 \le i \le r$, $1 \le j \le n$, such that $\mathcal{L}(E) = L$; that is, $w = b_1 \dots b_k \in L$ if and only if $w \in \mathcal{L}(E)$. Set $V = \{S, A, 0, 1, \$\} \cup T$. Define the **SCG** $G = (V, T, P, S)$ with P constructed as follows:

1. For every $a \in T$, add
   a) $(S) \to ((z_a)^R Sa)$, and
   b) $(S) \to ((z_a)^R Aa)$ to P;
2. a) For every $(u_i, v_i) \in E$, $1 \le i \le r$, add $(A) \to ((u_i)^R A v_i)$ to P;
   b) Add $(A) \to (\$\$)$ to P;
3. Add
   a) $(0, \$, \$, 0) \to (\$, \varepsilon, \varepsilon, \$)$,
   b) $(1, \$, \$, 1) \to (\$, \varepsilon, \varepsilon, \$)$, and
   c) $(\$) \to (\varepsilon)$ to P.

Claim 3. *Let $w_1, w_2 \in \{0,1\}^*$. Then, $w_1\$\$w_2 \Rightarrow_G^* \varepsilon$ if and only if $w_1 = (w_2)^R$.*

Proof. If: Let $w_1 = (w_2)^R = b_1 \dots b_k$, for some $k \geq 0$. By productions (3a) and (3b) followed by two applications of (3c), we obtain

$$\begin{aligned} b_k \dots b_2 b_1 \$\$ b_1 b_2 \dots b_k &\Rightarrow b_k \dots b_2 \$\$ b_2 \dots b_k \\ &\Rightarrow^* b_k \$\$ b_k \\ &\Rightarrow \$\$ \Rightarrow \$ \Rightarrow \varepsilon. \end{aligned}$$

Therefore the if-part of the claim holds.

Only if: Suppose that $|w_1| \leq |w_2|$. We demonstrate that

$$w_1\$\$w_2 \Rightarrow_G^* \varepsilon \text{ implies } w_1 = (w_2)^R$$

by induction on $k = |w_1|$.

Basis: Let $k = 0$. Then, $w_1 = \varepsilon$ and the only possible derivation is

$$\$\$w_2 \Rightarrow \$w_2 \ [(3c)] \Rightarrow w_2 \ [(3c)].$$

Hence, we can derive ε only if $w_1 = (w_2)^R = \varepsilon$.

Induction Hypothesis: Suppose that the claim holds for all w_1 satisfying $|w_1| < k$ for some $k \geq 0$.

Induction Step: Consider $w_1 a\$\$bw_2$ with $a \neq b$, $a, b \in \{0,1\}$. If $w_1 = w_{11} b w_{12}$, $w_{11}, w_{12} \in \{0,1\}^*$, then either (3a) or (3b) can be used. In either case, we obtain

$$w_1 a\$\$bw_2 \Rightarrow w_{11}\$w_{12}aw_{21}\$w_{22},$$

where $bw_2 = w_{21} b w_{22}$, $w_{21}, w_{22} \in \{0,1\}^*$, and $w_{12}aw_{21} \in N^+$ cannot be removed by any production from the sentential form. The same is true when $w_2 = w'_{21} a w'_{22}$, $w'_{21}, w'_{22} \in \{0,1\}^*$. Therefore, the derivation proceeds successfully only if $a = b$. Thus,

$$w_1 a\$\$bw_2 \Rightarrow w_1\$\$w_2 \Rightarrow^* \varepsilon,$$

and from the induction hypothesis,

$$w_1 = (w_2)^R.$$

Analogously, the same result can be proved for $|w_1| \geq |w_2|$, which implies that the only-if part of the claim holds.

Therefore, the claim holds. Δ

Examine the introduced productions to see that G always generates $b_1 \dots b_k \in \mathcal{L}(E)$ by a derivation of this form:

$$\begin{aligned}
S &\Rightarrow (z_{b_k})^R S b_k \\
&\Rightarrow (z_{b_k})^R (z_{b_{k-1}})^R S b_{k-1} b_k \\
&\Rightarrow^* (z_{b_k})^R \dots (z_{b_2})^R S b_2 \dots b_k \\
&\Rightarrow (z_{b_k})^R \dots (z_{b_2})^R (z_{b_1})^R A b_1 b_2 \dots b_k \\
&\Rightarrow (z_{b_k})^R \dots (z_{b_1})^R (u_{s_l})^R A v_{s_l} b_1 \dots b_k \\
&\Rightarrow^* (z_{b_k})^R \dots (z_{b_1})^R (u_{s_l})^R \dots (u_{s_1})^R A v_{s_1} \dots v_{s_l} b_1 \dots b_k \\
&\Rightarrow (z_{b_k})^R \dots (z_{b_1})^R (u_{s_l})^R \dots (u_{s_1})^R \$\$ v_{s_1} \dots v_{s_l} b_1 \dots b_k \\
&\quad = (u_{s_1} \dots u_{s_l} z_{b_1} \dots z_{b_k})^R \$\$ v_{s_1} \dots v_{s_l} b_1 \dots b_k \\
&\Rightarrow^* b_1 \dots b_k.
\end{aligned}$$

Productions introduced in steps 1 and 2 of the construction find nondeterministically the solution of the extended Post correspondence problem which is subsequently verified by productions from step 3. Therefore, $w \in L$ if and only if $w \in \mathcal{L}(G)$ and the theorem holds. □

Theorem 3. *Let m and h be positive integers. Then, ${}_mCF = {}_{nonter}SC(m,h)$.*

Proof. Obviously, ${}_mCF \subseteq {}_{nonter}SC(m,h)$.

We prove that ${}_{nonter}SC(m,h) \subseteq {}_mCF$. Let $\alpha = x_0 y_1 x_1 \dots y_n x_n$, where $x_i \in T^*$, $y_i \in N^+$, for $0 \leq i \leq n$, and for all $0 < i < n$, $x_i \neq \varepsilon$. Define $f(\alpha) = x_0 \langle y_1 \rangle x_1 \dots \langle y_n \rangle x_n$, where $\langle y_i \rangle$ is a new nonterminal, for all $0 \leq i \leq n$. Let $G_{SC} = (V, T, P, S)$ be a generalized scattered context grammar. Introduce a context-free grammar $G_{CF} = (V', T, P', \langle S \rangle)$, where $V' = \{\langle \gamma \rangle : \gamma \in N^*, 1 \leq |\gamma| \leq h\} \cup T$ and P' is constructed as follows:

1. for each $\gamma = x_0 \alpha_1 x_1 \dots \alpha_n x_n$, where $x_i \in N^*$, $\alpha_i \in N^+$, $1 \leq |\gamma| \leq h$, and $(\alpha_1, \dots, \alpha_n) \to (\beta_1, \dots, \beta_n) \in P$, add $\langle \gamma \rangle \to f(x_0 \beta_1 x_1 \dots \beta_n x_n)$ to P'.

Claim 4. *Let $S \overset{h}{\underset{m}{\Rightarrow}}{}^k \omega$ in G_{SC}, where $\omega \in V^*$, $k \geq 0$. Then, $\langle S \rangle \; {}_m\Rightarrow^k f(\omega)$ in G_{CF}.*

Proof. By induction on $k = 0, 1, \dots$.

Basis: Let $k = 0$, thus $S \overset{h}{\underset{m}{\Rightarrow}}{}^0 S$ in G_{SC}. Then, $\langle S \rangle \; {}_m\Rightarrow^0 \langle S \rangle$ in G_{CF}. As $f(S) = \langle S \rangle$, the basis holds.

Induction hypothesis: Suppose that the claim holds for all $0 \leq m \leq k$, where k is a non-negative integer.

Induction step: Let $S \overset{h}{\underset{m}{\Rightarrow}}{}^k \phi\gamma\psi \overset{h}{\underset{m}{\Rightarrow}} \phi\gamma'\psi$ in G_{SC}, and the last production applied during the derivation is $(\alpha_1, \dots, \alpha_n) \to (\beta_1, \dots, \beta_n)$, where $\phi \in V^*T \cup \{\varepsilon\}$, $\gamma = x_0 \alpha_1 x_1 \dots \alpha_n x_n$, $\psi \in TV^* \cup \{\varepsilon\}$, $\gamma' = x_0 \beta_1 x_1 \dots \beta_n x_n$, $\alpha_i, x_i \in N^*$, and $\beta_i \in V^*$. By the induction hypothesis,

$$\langle S \rangle \; {}_m\Rightarrow^k f(\phi\gamma\psi).$$

By the definition of f, ϕ, and ψ, $f(\phi\gamma\psi) = f(\phi)\langle\gamma\rangle f(\psi)$. Hence, we can use the production $\langle\gamma\rangle \to f(\gamma') \in P'$ introduced in 1 in the construction to obtain

$$f(\phi)\langle\gamma\rangle f(\psi) \;{}_m\Rightarrow\; f(\phi)f(\gamma')f(\psi).$$

By the definition of f, ϕ, and ψ, $f(\phi)f(\gamma')f(\psi) = f(\phi\gamma'\psi)$. As a result,

$$\langle S\rangle \;{}_m\Rightarrow^k f(\phi)\langle\gamma\rangle f(\psi) \;{}_m\Rightarrow\; f(\phi\gamma'\psi)$$

and, therefore, $\langle S\rangle \;{}_m\Rightarrow^{k+1} f(\phi\gamma'\psi)$ and the claim holds for $k+1$. △

Claim 5. *Let* $\langle S\rangle \;{}_m\Rightarrow^k \omega$ *in* G_{CF}, *where* $\omega \in V'^*$, $k \geq 0$. *Then,* $S \;{}^h_m\!\!\Rightarrow^k f^{-1}(\omega)$ *in* G_{SC}.

Proof. By induction on $k = 0, 1, \ldots$.

Basis: Let $k = 0$, thus $\langle S\rangle \;{}_m\Rightarrow^0 \langle S\rangle$ in G_{CF}. Then $S \;{}^h_m\!\!\Rightarrow^0 S$ in G_{SC}. As $f^{-1}(\langle S\rangle) = S$, the basis holds.

Induction hypothesis: Suppose that the claim holds for all $0 \leq m \leq k$, where k is a non-negative integer.

Induction step: Let $\langle S\rangle \;{}_m\Rightarrow^k \phi\langle\gamma\rangle\psi \;{}_m\Rightarrow\; \phi\gamma'\psi$ in G_{CF}, and the last production applied during the derivation is $\langle\gamma\rangle \to \gamma'$, where $\phi \in V^*T \cup \{\varepsilon\}$, $\gamma = x_0\alpha_1x_1 \ldots \alpha_nx_n$, $\psi \in TV^* \cup \{\varepsilon\}$, $\gamma' = f(x_0\beta_1x_1 \ldots \beta_nx_n)$, $\alpha_i, x_i \in N^*$, and $\beta_i \in V^*$. By the induction hypothesis,

$$S \;{}^h_m\!\!\Rightarrow^k f^{-1}(\phi\langle\gamma\rangle\psi).$$

By the definition of f, ϕ, and ψ, $f^{-1}(\phi\langle\gamma\rangle\psi) = f^{-1}(\phi)\gamma f^{-1}(\psi)$. There exists $(\alpha_1, \ldots, \alpha_n) \to (\beta_1, \ldots, \beta_n) \in P$ by 1, thus

$$f^{-1}(\phi)\gamma f^{-1}(\psi) \;{}^h_m\!\!\Rightarrow\; f^{-1}(\phi)f^{-1}(\gamma')f^{-1}(\psi).$$

By the definition of f, ϕ, and ψ, $f^{-1}(\phi)f^{-1}(\gamma')f^{-1}(\psi) = f^{-1}(\phi\gamma'\psi)$. As a result

$$S \;{}^h_m\!\!\Rightarrow^k f^{-1}(\phi)\gamma f^{-1}(\psi) \;{}^h_m\!\!\Rightarrow\; f^{-1}(\phi\gamma'\psi)$$

and, therefore, $S \;{}^h_m\!\!\Rightarrow^{k+1} f^{-1}(\phi\gamma'\psi)$ and the claim holds for $k+1$. △

Hence, the theorem holds. □

References

[1] Baker, B. S. Context-sesitive grammars generating context-free languages. In Nivat, M., editor, *Automata, Languages and Programming*, pages 501–506. North-Holland, Amsterdam, 1972.

[2] Book, R. V. Terminal context in context-sensitive grammars. *SIAM Journal of Computing*, 1:20–30, 1972.

[3] Geffert, V. Context-free-like forms for the phrase-structure grammars. In Chytil, M., Janiga, L., and Koubek, V., editors, *Mathematical Foundations of Computer Science*, volume 324 of *Lecture Notes in Computer Science*, pages 309–317. Springer-Verlag, 1988.

[4] Ginsburg, S. and Greibach, S. Mappings which preserve context-sensitive languages. *Information and Control*, 9:563–582, 1966.

[5] Greibach, S. and Hopcroft, J. Scattered context grammars. *Journal of Computer and System Sciences*, 3:233–247, 1969.

[6] Matthews, G. A note on symmetry in phrase structure grammars. *Information and Control*, 7:360–365, 1964.

[7] Matthews, G. Two-way languages. *Information and Control*, 10:111–119, 1967.

[8] Meduna, A. A trivial method of characterizing the family of recursively enumerable languages by scattered context grammars. *EATCS Bulletin*, pages 104–106, 1995.

[9] Rozenberg, G. and Salomaa, A., editors. *Handbook of Formal Languages*, volume 1. Springer-Verlag, Berlin, 1997.

[10] Salomaa, A. *Formal languages.* Academic Press, New York, 1973.

Received 18th July 2007

Acta Cybernetica **18** (2008) 795–805.

Partially Ordered Pattern Algebras

Endre Vármonostory*

Abstract

A partial order $\preceq$ on a set A induces a partition of each power A^n into "patterns" in a natural way. An operation on A is called a $\preceq$-pattern operation if its restriction to each pattern is a projection. We examine functional completeness of algebras with $\preceq$-pattern fundamental operations.

Keywords: majority function, semiprojection, ternary discriminator, dual discriminator, functionally completeness

1 Preliminaries

A finite algebra $\mathbf{A} = (A; F)$ is called *functionally complete* if every (finitary) operation on A is a polinomial operation of $\mathbf{A}$. An n-ary operation f on A is *conservative* if $f(x_1, \ldots, x_n) \in \{x_1, \ldots, x_n\}$ for all $x_1, \ldots, x_n \in A$. An algebra is conservative if all of its fundamental operations are conservative.

A possible approach to the study of conservative operations is to consider them as relational pattern functions or ρ-pattern functions. Given a k-ary relation $\rho \subseteq A^k$, two n-tuples $(x_1, \ldots, x_n)$, $(y_1, \ldots, y_n) \in A^n$ are said to be of the same pattern with respect to ρ if for all $i_1, \ldots, i_k \in \{1, \ldots, n\}$ the conditions $(x_{i_1}, \ldots, x_{i_k}) \in \rho$ and $(y_{i_1}, \ldots, y_{i_k}) \in \rho$ mutually imply each other. An operation $f : A^n \to A$ is a *ρ-pattern function* if $f(x_1, \ldots, x_n)$ always equals some x_i, $i \in \{1, \ldots, n\}$ where i depends only on the ρ-pattern of $(x_1, \ldots, x_n)$. In fact, any conservative operation is a ρ-pattern function for some ρ — see [11]. An algebra $\mathbf{A}$ is called a *ρ-pattern algebra* if its fundamental operations (or equivalently its term operations) are ρ-pattern functions for the same relation ρ on A. Several facts about functional completeness were proved, for the cases where ρ is an equivalence [9], a central relation [10, 14], a graph of a permutation [13], a bounded partial order [12], or a regular relation [8] on A. These relations appear in Rosenberg's primality criterion [6].

In particular if $\preceq$ is a partial order or a linear order on A, then a $\preceq$-pattern algebra is called a partially ordered pattern algebra or a linearly ordered pattern algebra. Throughout the paper such algebras will be called *$\preceq$-pattern algebras.*

*University of Szeged, Gyula Juhász Faculty of Education, Hattyas sor 10., H-6725 Szeged, Hungary. E-mail: varmono@jgypk.u-szeged.hu

The aim of this article is to continue research on functional completeness of finite partially ordered pattern algebras.

In case when the relation ρ on A is the identity the ρ-pattern algebra is called *pattern algebra*. The basic operations of pattern algebras are called pattern functions. Pattern functions were first introduced by Quackenbush [5]. B. Csákány [1] proved that every finite pattern algebra $(A; f)$ with $|A| \geq 3$ is functionally complete if f is an arbitrary nontrivial pattern function. The most known examples of pattern algebras are $(A; f)$ and $(A; g)$ where f is the *ternary discriminator* [4] ($f(x, y, z) = z$ if $x = y$ and $f(x, y, z) = x$ if $x \neq y$) and g is the *dual discriminator* [2] ($g(x, y, z) = x$ if $x = y$ and $g(x, y, z) = z$ if $x \neq y$).
We need the following definitions and results.

An n-ary relation ρ on A is called *central* iff $\rho \neq A^n$ and

(a) there exists $c \in A$ such that $(a_1, \ldots, a_n) \in \rho$ whenever at least one $a_i = c$ (the set of all such c's is called the *center* of ρ);

(b) $(a_1, \ldots, a_n) \in \rho$ implies that $(a_{1\pi}, \ldots, a_{n\pi}) \in \rho$ for every permutation π of $\{1, \ldots, n\}$ (ρ is *totally symmetric*);

(c) $(a_1, \ldots, a_n) \in \rho$ whenever $a_i = a_j$ for some $i \neq j$ (ρ is *totally reflexive*).

Let A be a finite and nonempty set, $k, n \geq 1$, f a k-ary function on A and $\rho \subseteq A^n$ an arbitrary n-ary relation. The operation f is said to *preserve* ρ if ρ is a subalgebra of the nth direct power of the algebra $(A; f)$; in other words, f preserves ρ if for any $k \times n$ matrix M with entries in A, whose rows belong to ρ, the row obtained by applying f to the columns of M also belongs to ρ. Adding this extra row to M we get a so-called f-matrix [3].

A ternary operation f on A is a *majority function* if $f(x, x, y) = f(x, y, x) = f(y, x, x) = x$ holds for all $x, y \in A$. An n-ary i-th *semiprojection* on A ($n \geq 3$, $1 \leq i \leq n$) is an operation f with the property that $f(x_1, x_2, \ldots, x_n) = x_i$ whenever at least two of the elements $x_1, \ldots, x_n$ are equal. The following proposition was obtained in [13] from Rosenberg's fundamental theorem on minimal clones [7].

Proposition 1. *The clone of the term operations of every nontrivial finite ρ-pattern algebra* **A** *with at least three elements contains a nontrivial binary ρ-pattern function, or a ternary majority ρ-pattern function, or a nontrivial ρ-pattern function, which is a semiprojection.*

Now we formulate the following theorem (which was got from Proposition 4 in [13]).

Theorem 2. *Let* $\mathbf{A} = (A; f)$ *be a finite ρ-pattern algebra with* $|A| \geq 3$. *The algebra* $(A; f)$ *is functionally complete iff*

(a) f is monotonic with respect to no bounded partial order on A,

(b) f preserves no binary central relations on A,

(c) f preserves no nontrivial equivalences on A.

2 Results

Theorem 3. *Let $(A;\preceq)$ be a finite poset with at least three elements that has a least or a greatest element. If f is an arbitrary binary $\preceq$-pattern function on A, then the algebra $(A;f)$ is not functionally complete.*

Proof. Let a be the least or the greatest element of $(A;\preceq)$. Let ρ be the nontrivial equivalence on A with blocks $\{a\}$, $A\backslash\{a\}$. Now f preserves ρ and apply Theorem 2. □

Remark. Let $\underline{n}=\{0,1,\dots,n-1\}$ be an at least three-element set, and let $\preceq$ be a linear order on $\underline{n}$ such that $0\preceq i\preceq n-1$ holds for each $i\in\underline{n}$. If $a,b\in\underline{n}$ and $a\preceq b$ but $a\neq b$ then we write $a\prec b$. Now the following statement is true.

If π and σ are two different permutations of the set $\{1,2,\dots,k\}$ then the k-tuples $(a_{1\pi},a_{2\pi},\dots,a_{k\pi})$, $(a_{1\sigma},a_{2\sigma},\dots,a_{k\sigma})$ are not in the same pattern with respect to $\preceq$ where $a_1,a_2,\dots,a_k\in\underline{n}$ with $a_1\prec a_2\prec\dots\prec a_k$.

Now we can formulate the following theorem.

Theorem 4. *Let $(A;\preceq)$ be a finite linearly ordered set with $|A|=n\geq 4$, and let f be a $\preceq$-pattern function that is a majority function on A. Then the algebra $(A;f)$ is functionally complete iff for arbitrary elements $a_1,a_2,a_3\in A$ with $a_1\prec a_2\prec a_3$ exactly one of the following statements holds:*

(a) there exist permutations π,σ of the set $\{1,2,3\}$ for which the values $f(a_1,a_2,a_3)$, $f(a_{1\pi},a_{2\pi},a_{3\pi})$, $f(a_{1\sigma},a_{2\sigma},a_{3\sigma})$ are pairwise distinct,

(b) $f(a_{1\pi},a_{2\pi},a_{3\pi})\in\{a_1,a_3\}$ for every permutation π of $\{1,2,3\}$, and there exists a permutation π' of $\{1,2,3\}$ for which $f(a_{1\pi'},a_{2\pi'},a_{3\pi'})\neq f(a_1,a_2,a_3)$.

Proof. We will use Theorem 2. We may suppose, without loss of generality, that $A=\underline{n}$. First, we prove that if one of the conditions (a) or (b) hold for the algebra $(\underline{n};f)$ then f preserves neither the bounded partial orders nor the binary central relations on $\underline{n}$. We need the following claims.

Claim. *Let $\trianglelefteq$ be an arbitrary bounded partial order on $\underline{n}$ with the least element m and the greatest element M, then f does not preserve $\trianglelefteq$.*

Proof of Claim. If $a\in\underline{n}$, $a\neq m$, M, then $f(m,a,M)=m$ or $f(m,a,M)=M$ or $f(m,a,M)=a$. Consider the following f-matrices

m	m	m	a
a	a	a	a
a	M	M	M
$f(m,a,a)$	$f(m,a,M)$	$f(m,a,M)$	$f(a,a,M)$

where $f(m,a,a)=f(a,a,M)=a$. If $f(m,a,M)=m$, then the first f-matrix shows that f does not preserve $\trianglelefteq$. If $f(m,a,M)=M$, then by the second f-matrix f does not preserve $\trianglelefteq$. If $f(m,a,M)=a$, then by (a) or (b) we get that at least

one of the elements $f(m, M, a)$, $f(M, m, a)$, $f(M, a, m)$, $f(a, m, M)$, $f(a, M, m)$ is equal to m or M. In this case we can get the suitable f-matrix by permuting the first three rows of one of the two f-matrices above. Now from this f-matrix we get that f does not preserve $\trianglelefteq$. The proof of the claim is complete.

Claim. *If τ is an arbitrary binary central relation on $\underline{n}$, then f does not preserve τ.*

Proof of Claim. If $c \in \underline{n}$ is a central element of τ and $a, b \in \underline{n}$ so that $(a, b) \notin \tau$, then consider the following matrices

$$\begin{array}{cc} a & a \\ b & b \\ c & b \\ \hline f(a,b,c) & f(a,b,b) \end{array} \qquad \begin{array}{cc} a & a \\ b & b \\ c & a \\ \hline f(a,b,c) & f(a,b,a) \end{array}$$

where $f(a, b, b) = b$ and $f(a, b, a) = a$. If $f(a, b, c) = a$, then the first f-matrix shows that f does not preserve τ. If $f(a, b, c) = b$, then the second f-matrix will be used. If $f(a, b, c) = c$, then by (a) or (b) we see that $f(a, c, b)$, $f(b, a, c)$, $f(b, c, a)$, $f(c, a, b)$ or $f(c, b, a)$ is equal to a or b. Now we can also get the suitable f-matrix by permuting the first three rows of one of the two f-matrices above. In this case from this f-matrix we get that f does not preserve τ. The proof of the claim is complete.

Now we will prove that if one of the conditions (a) or (b) holds for the algebra $(\underline{n}; f)$, then f does not preserve the nontrivial equivalences on $\underline{n}$.

Claim. *If ρ is an arbitrary nontrivial equivalence on $\underline{n}$, then f does not preserve ρ.*

Proof of Claim. Now there exist elements a, b, $c \in \underline{n}$ with $a \neq b$, $(a, b) \in \rho$, $(a, c) \notin \rho$.
First, suppose that (a) holds. If $f(a, b, c) = c$, then we can use the following f-matrix to show that f does not preserve ρ

$$\begin{array}{cc} a & a \\ a & b \\ c & c \\ \hline a & c \end{array}$$

where $f(a, a, c) = a$. If $f(a, b, c) = a$ or $f(a, b, c) = b$, then by (a) $f(a, c, b)$, $f(b, a, c)$, $f(b, c, a)$, $f(c, a, b)$ or $f(c, a, b)$ equals c. In this case we get the suitable f-matrix by permuting the first three rows of the f-matrix above. From this f-matrix we get that f does not preserve ρ.

Now we suppose that (b) is true.

(i) First, we suppose that $a \prec b \prec c$. If $f(a,b,c) = c$, then the f-matrix above does the job. If $f(a,b,c) = a$, then by (b) $f(a,c,b)$, $f(b,a,c)$, $f(b,c,a)$, $f(c,a,b)$ or $f(c,b,a)$ equals c. We get the suitable f-matrix by permuting the first three rows of the f-matrix above.

(ii) Secondly, we suppose that $c \prec a \prec b$. If $f(c,a,b) = c$ then we get the suitable f-matrix by permuting the first three rows of the f-matrix above. If $f(c,a,b) = b$, then by (b) $f(c,b,a)$, $f(a,b,c)$, $f(a,c,b)$, $f(b,a,c)$, $f(b,c,a)$ equals c. For example, if $f(c,b,a) = c$, then the following f-matrix shows that f does not preserve ρ

$$\begin{array}{cc} c & c \\ b & a \\ a & a \\ \hline c & a \end{array}.$$

In the remaining cases we get the suitable f-matrices by permuting the first three rows of the f-matrix above.

(iii) If there do not exist elements $a,b,c \in \underline{n}$ with $a \neq b$, $(a,b) \in \rho$, $(a,c) \notin \rho$ for which $a \prec b \prec c$ or $c \prec a \prec b$ hold, then it is easy to see that ρ has a unique nonsingleton block, namely $\{0, n-1\}$. Now $|A| \geq 4$ and we can suppose that $a = 0$, $b = n-1$ and $\{c_1, \ldots, c_{n-2}\} = \underline{n} \setminus \{a,b\}$.

First, assume $f(a,c_1,c_2) = a$. If $f(b,c_1,c_2) = c_1$, then the following f-matrix

$$\begin{array}{cc} a & b \\ c_1 & c_1 \\ c_2 & c_2 \\ \hline a & c_1 \end{array}$$

will be used. If $f(b,c_1,c_2) = b$, then $f(c_2,a,c_1) = c_2$ since the patterns (b,c_1,c_2) and (c_2,a,c_1) are the same with respect to $\preceq$. We need the following f-matrices

$$\begin{array}{cc} c_2 & c_2 \\ a & b \\ c_1 & c_1 \\ \hline c_2 & c_1 \end{array} \qquad \begin{array}{cc} c_2 & c_2 \\ a & b \\ c_1 & c_1 \\ \hline c_2 & b \end{array}.$$

If $f(c_2,b,c_1) = c_1$, then the first f-matrix shows that f does not preserve ρ. If $f(c_2,b,c_1) = b$, then the second f-matrix does the job.

Secondly, assume $f(a, c_1, c_2) = c_2$. Now we will use the following f-matrices

$$\begin{array}{cc} a & b \\ c_1 & c_1 \\ c_2 & c_2 \\ \hline c_2 & c_1 \end{array} \qquad \begin{array}{cc} a & b \\ c_1 & c_1 \\ c_2 & c_2 \\ \hline c_2 & b \end{array}.$$

If $f(b, c_1, c_2) = c_1$, then the first f-matrix shows that f does not preserve ρ.
If $f(b, c_1, c_2) = b$, then the second f-matrix will be used.

The proof of the claim is complete.

From now we show that the algebra $(\underline{n}; f)$ is not functionally complete if (a) and (b) are not satisfied. Further also suppose that $a_1, a_2, a_3 \in \underline{n}$ and $a_1 \prec a_2 \prec a_3$. We have the following three cases:

If $a_i = f(a_1, a_2, a_3) = f(a_{1\pi}, a_{2\pi}, a_{3\pi})$ equalities hold for every permutation π of $\{1, 2, 3\}$, then f preserves one of the three binary central relations τ_1, τ_2, τ_3 on A defined below:

For $i = 1$, let the center of τ_1 be $C = \{0, 1, \ldots, n-3\}$ and $(n-2, n-1) \notin \tau_1$,
for $i = 2$, let the center of τ_2 be $C = \{1, 2, \ldots, n-2\}$ and $(0, n-1) \notin \tau_2$,
for $i = 3$, let the center of τ_3 be $C = \{2, 3, \ldots, n-1\}$ and $(0, 1) \notin \tau_3$.

Now let $f(a_{1\pi}, a_{2\pi}, a_{3\pi}) \in \{a_1, a_2\}$ be for every permutation π of $\{1, 2, 3\}$ (or let $f(a_{1\pi}, a_{2\pi}, a_{3\pi}) \in \{a_2, a_3\}$ be for every permutation π of $\{1, 2, 3\}$), and suppose that there exists a permutation π' of $\{1, 2, 3\}$ for which $f(a_{1\pi'}, a_{2\pi'}, a_{3\pi'}) \neq f(a_1, a_2, a_3)$. Then it is easy to show that f preserves the nontrivial equivalence with a unique nonsingleton block, namely $\{0, 1, \ldots, n-2\}$ (or $\{1, 2, \ldots, n-1\}$).

□

Proposition 5. *Let $A = \{0, 1, 2\}$ be a linearly ordered set with $0 \prec 1 \prec 2$, and let f be a $\preceq$-pattern function, which is a majority function on A. Then the algebra $(A; f)$ is functionally complete iff there exist permutations π, σ of A for which the values $f(0, 1, 2)$, $f(0\pi, 1\pi, 2\pi)$, $f(0\sigma, 1\sigma, 2\sigma)$ are pairwise distinct.*

Proof. Suppose that there exist permutations π, σ of A for which the values $f(0, 1, 2)$, $f(0\pi, 1\pi, 2\pi)$, $f(0\sigma, 1\sigma, 2\sigma)$ are pairwise distinct. Then the algebra $(A; f)$ is functionally complete. (Let us observe that the proof of this statement is included in the proof of Theorem 4, since in the case (a) of Theorem 4 every f-matrix has exactly three elements.)

If $f(0, 1, 2) = f(0\pi, 1\pi, 2\pi)$ for every permutation π of A, then we obtain that f preserves one of the three binary central relations τ_1, τ_2, τ_3 on A defined below:

For $f(0, 1, 2) = 0$ let the center of τ_1 be $\{0\}$, and $(1, 2) \notin \tau_1$,
for $f(0, 1, 2) = 1$ let the center of τ_2 be $\{1\}$, and $(0, 2) \notin \tau_2$,
for $f(0, 1, 2) = 2$ let the center of τ_3 be $\{2\}$, and $(0, 1) \notin \tau_3$.

Now let assume that at least one of the inclusions: $f(0\pi, 1\pi, 2\pi) \in \{0,1\}$, $f(0\pi, 1\pi, 2\pi) \in \{1,2\}$, $f(0\pi, 1\pi, 2\pi) \in \{0,2\}$ holds for every permutation π of A, and suppose that there exists a permutation π' of A for which $f(0\pi', 1\pi', 2\pi') \neq f(0,1,2)$. Then it is also easy to observe that f preserves the nontrivial equivalence with unique nonsingleton block, namely $\{0,1\}$, $\{1,2\}$ or $\{0,2\}$. Using Theorem 2, the proof is complete. □

Theorem 6. *Let $(A, \preceq)$ be an arbitrary finite poset with $3 \leq |A|$. Let f be a $\preceq$-pattern function, which is a majority function on A, and for which there exist permutations π, σ of $\{1,2,3\}$ such that the values $f(a_1, a_2, a_3)$, $f(a_{1\pi}, a_{2\pi}, a_{3\pi})$, $f(a_{1\sigma}, a_{2\sigma}, a_{3\sigma})$ are pairwise distinct, then the algebra $(A; f)$ is functionally complete.*

Proof. Such an operation f always exists. (For example: $f(x,x,y) = f(x,y,x) = f(y,x,x) = x$, and $f(x,y,z) = x$ if x, y, z are pairwise different). Now it is easy to prove that such operations do not preserve the bounded partial orders, the binary central relations and the nontrivial equivalences on A. Applying Theorem 2, the proof is complete. □

Theorem 7. *Let $(A; \preceq)$ be an arbitrary finite poset with $3 \leq |A|$. Then for every k with $3 \leq k \leq |A|$ there exists a k-ary $\preceq$-pattern function f, which is a semiprojection and the algebra $(A; f)$ is functionally complete.*

Proof. If $3 \leq k \leq |A|$, then the k-ary $\preceq$-pattern function

$$f_k(x_1, x_2, \ldots, x_k) = \begin{cases} x_1 & \text{if the elements } x_1, x_2, \ldots, x_k \text{ are pairwise distinct and} \\ & x_{k-1} \not\prec x_k, \\ x_k & \text{otherwise} \end{cases}$$

is a semiprojection on A. By Lemma 7 of [3] f_k has no compatible bounded partial order on A.

Let τ be an arbitrary binary central relation on A, let $c \in A$ be a central element of τ, and let $a, b \in A$ be so that $(a,b) \notin \tau$. We will need the following matrices

$$\begin{array}{cc} a & a \\ d & d \\ \vdots & \vdots \\ e & e \\ c & b \\ b & b \\ \hline a & b \end{array} \qquad \begin{array}{cc} a & a \\ d & d \\ \vdots & \vdots \\ e & e \\ b & b \\ c & b \\ \hline a & b \end{array}$$

where the entries above the line in the first column are pairwise distinct in both f_k-matrices.

If $c \not\prec b$, then we will use the first f_k-matrix. If $c \prec b$, then the second f_k-matrix will work. In both cases we get that f_k does not preserve the relation τ.

Let ρ be an arbitrary nontrivial equivalence, and let $a, b, c \in A$ with $a \not\prec b$, $(a, b) \in \rho$ and $(a, c) \notin \rho$. Now we will use the following f_k-matrix to show that f_k does not preserve ρ

$$\begin{array}{cc} c & c \\ d & d \\ \vdots & \vdots \\ e & e \\ a & a \\ \underline{b} & \underline{a} \\ c & a \end{array}$$

where the entries above the line in the first column of the f_k-matrix are pairwise distinct. Using Theorem 2 we get that the algebra $(A; f_k)$ is functionally complete. □

Remark. Let $(A; \preceq)$ be a finite linearly ordered set with $3 \leq |A|$, and let f be a nontrivial k-ary $\preceq$-pattern function, which is a semiprojection on A. If for any elements $a_1, \ldots, a_k \in A$ with $a_1 \prec \ldots \prec a_k$, and for any permutations π of $\{1, \ldots, k\}$ one of the following conditions is satisfied:

(a) $a_i = f(a_{1\pi}, \ldots, a_{k\pi})$, $3 \leq k \leq |A|$, or

(b) $f(a_{1\pi}, \ldots, a_{k\pi}) \in \{a_1, a_2, \ldots, a_{k-2}\}$, $4 \leq k \leq |A|$, or

(c) $f(a_{1\pi}, \ldots, a_{k\pi}) \in \{a_2, a_3, \ldots, a_{k-1}\}$, $4 \leq k \leq |A|$, or

(d) $f(a_{1\pi}, \ldots, a_{k\pi}) \in \{a_3, a_4, \ldots, a_k\}$, $4 \leq k \leq |A|$

then the algebra $(A; f)$ is not functionally complete.

Proof of Remark. We may suppose, without loss of generality, that $A = \underline{n}$. If condition (a) holds, then f preserves one of the binary central relation τ_1, τ_2, τ_3 on A defined below:

(1) for $i = 1$, let the center of τ_1 be $C = \{0, 1, \ldots, n-3\}$ and $(n-2, n-1) \notin \tau_1$,

(2) for $1 < i < k$, let the center of τ_2 be $C = \{1, 2, \ldots, n-2\}$ and $(0, n-1) \notin \tau_2$,

(3) for $i = k$, let the center of τ_3 be $C = \{2, 3, \ldots, n-1\}$ and $(0, 1) \notin \tau_3$.

It is also easy to see that if (b) holds, then f preserves the central relation τ_1. If (c) (or (d)) holds, then f preserves the central relation τ_2 (or τ_3). Using Theorem 2, the proof of the remark is complete.

□

Let $(A;\preceq)$ be an arbitrary finite bounded poset with at least three elements. Define the following two operations on A:

$$t(x,y,z)=\begin{cases} z & if \quad x\preceq y,\\ x & \text{otherwise,}\end{cases}$$

$$d(x,y,z)=\begin{cases} x & if \quad x\preceq y,\\ z & \text{otherwise.}\end{cases}$$

The operation t is the *ternary order-discriminator*, and d is the *dual order-discriminator*. The algebras $(A;t)$, $(A;d)$ are called *order-discriminator algebras*. In [12] we proved that the order-discriminator algebras $(A;t)$ and $(A;d)$ are functionally complete. The following theorem is a generalization of this result.

Theorem 8. *If $(A;\preceq)$ is an arbitrary finite poset with at least three elements, then the order-discriminator algebras $(A;t)$ and $(A;d)$ are functionally complete.*

Proof. It is sufficient to prove that t and d do not preserve the relations (a), (b), and (c) in Theorem 2.

(a) Let $\trianglelefteq$ be an arbitrary bounded partial order on A with the least element m and the greatest element M. Now we show that the operations t, d do not preserve the bounded partial order $\trianglelefteq$ on A. Let $a\in A$ be an arbitrary element different from m and M. The following two t-matrices and two d-matrices will be used

$$\begin{array}{cc} m & m\\ m & a\\ M & M\\ \hline M & m\end{array}\qquad \begin{array}{cc} a & M\\ m & M\\ m & m\\ \hline a & m\end{array}\qquad \begin{array}{cc} a & M\\ a & a\\ m & m\\ \hline a & m\end{array}\qquad \begin{array}{cc} a & a\\ a & M\\ m & m\\ \hline a & m\end{array}.$$

If $a\prec m$ then the first t-matrix, if $a\not\prec m$ then the second t-matrix shows that t does not preserve $\trianglelefteq$. If $a\prec M$ then the first d-matrix, if $a\not\prec M$ then the second d-matrix shows that d does not preserve $\trianglelefteq$.

(b) Let τ be an arbitrary central relation on A, and let $a,b,c\in A$ so that $a\neq b$, $(a,b)\notin\tau$ and c is a central element of τ. We may suppose that $a\not\prec b$. Consider the following t-matrix and d-matrix

$$\begin{array}{cc} a & c\\ b & c\\ c & b\\ \hline a & b\end{array}\qquad \begin{array}{cc} a & a\\ a & c\\ c & b\\ \hline a & b\end{array}.$$

The first t-matrix shows that the operation t does not preserve τ. If $a\not\preceq c$ then by the d-matrix we see that the operation d does not preserve τ. If $a\preceq c$, then by permuting the first two rows of the d-matrix we get again that d does not preserve τ.

(c) Let ε be an arbitrary nontrivial equivalence on A, and let $a, b, c \in A$ so that $(a, b) \in \varepsilon$ and $(a, c) \notin \varepsilon$. We will need the following two t-matrices and two d-matrices:

$$\begin{array}{cc} a & b \\ a & a \\ c & c \\ \hline c & b \end{array} \qquad \begin{array}{cc} a & a \\ a & b \\ c & c \\ \hline c & a \end{array} \qquad \begin{array}{cc} a & b \\ a & a \\ c & c \\ \hline a & c \end{array} \qquad \begin{array}{cc} a & a \\ a & b \\ c & c \\ \hline a & c \end{array}.$$

If $a \prec b$, then by the first t-matrix, if $a \not\prec b$, then by the second t-matrix we get that the operation t does not preserve the relation ε. If $a \prec b$, then the first d-matrix, if $a \not\prec b$, then the second d-matrix does the job. In all cases we see that the operations t and d do not preserve ε. □

References

[1] Csákány, B. Homogeneous algebras are functionally complete. *Algebra Universalis*, 11:149–158, 1980.

[2] Fried, E. and Pixley, A.F. The dual discriminator function in universal algebra. *Acta Sci. Math.*, 41:83–100, 1979.

[3] Pálfy, P.P., Szabó, L., and Szendrei, Á. Automorphism groups and functional completeness. *Algebra Universalis*, 15:385–400, 1982.

[4] Pixley, A.F. The ternary discriminator function in universal algebra. *Math. Ann.*, 191:167–180, 1971.

[5] Quackenbush, R.W. Some classes of idempotent functions and their compositions. *Coll. Math. Soc. J. Bolyai*, 29:71–81, 1974.

[6] Rosenberg, G. Über die functionale Vollständigkeit in den mehrwertigen Logiken (Structur der Functionen von mehreren Veränderlichen auf eudlichen Mengen). *Rozpravy Československé Akad. Věd. Řada Math. Přírod. Věd.*, 80:3–93, 1970.

[7] Rosenberg, G. Minimal clones I: The five types. *Coll. Math. Soc. J. Bolyai*, 43:635–652, 1981.

[8] Szabó, L. and Vármonostory, E. On characteristically simple conservative algebras. *Publicationes Mathematicae*, 57:425–433, 2000.

[9] Vármonostory, E. Relational pattern functions. *Finite Algebra and Multiple-valued Logic (Proc. Conf. Szeged, 1979), Coll. Math. Soc. Bolyai*, 28:753–758, 1981.

[10] Vármonostory, E. Central pattern functions. *Acta Sci. Math.*, 56:223–227, 1992.

[11] Vármonostory, E. Generalized pattern functions. *Algebra Universalis*, 29:346–353, 1992.

[12] Vármonostory, E. Order-discriminating operations. *Order*, 9:239–244, 1992.

[13] Vármonostory, E. Permutation-pattern algebras. *Algebra Universalis*, 45:435–448, 2001.

[14] Vármonostory, E. Totally reflexive, totally symmetric pattern algebras. *Mathematica*, 47(72)(2):223–300, 2005.

Received 19th February 2007

CONTENTS

ISSN 0324—721 X

Felelős szerkesztő és kiadó: Csirik János